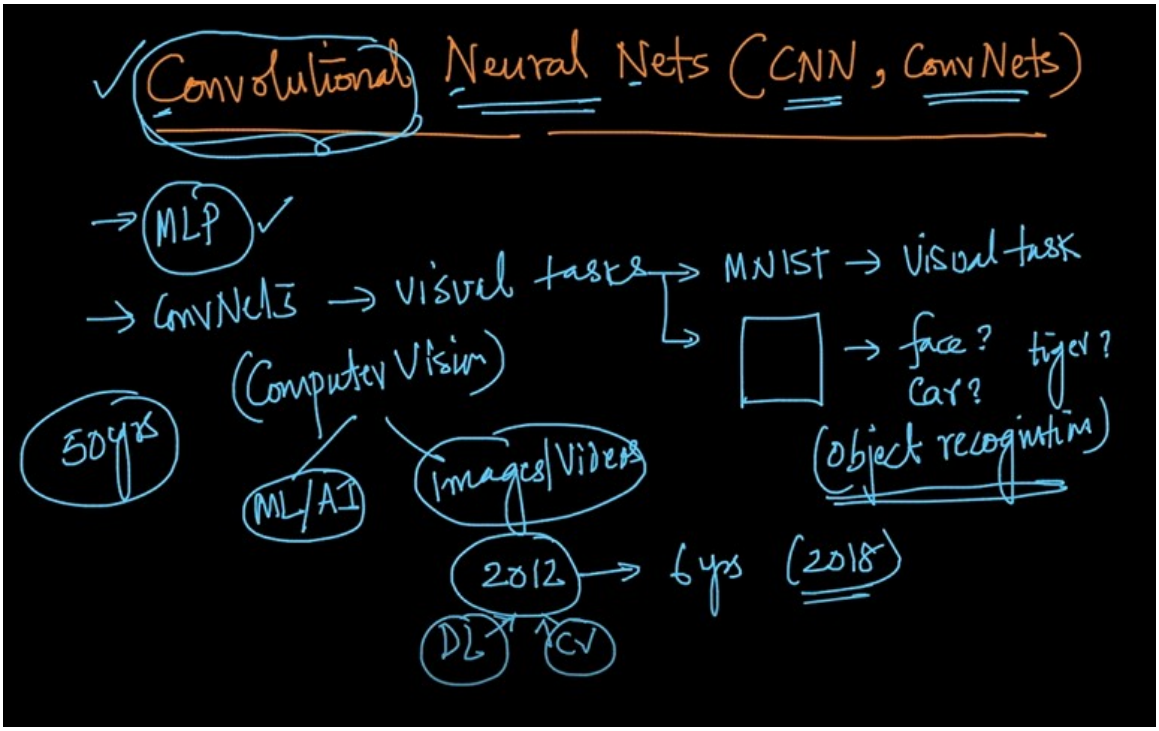
Convolutional neural network

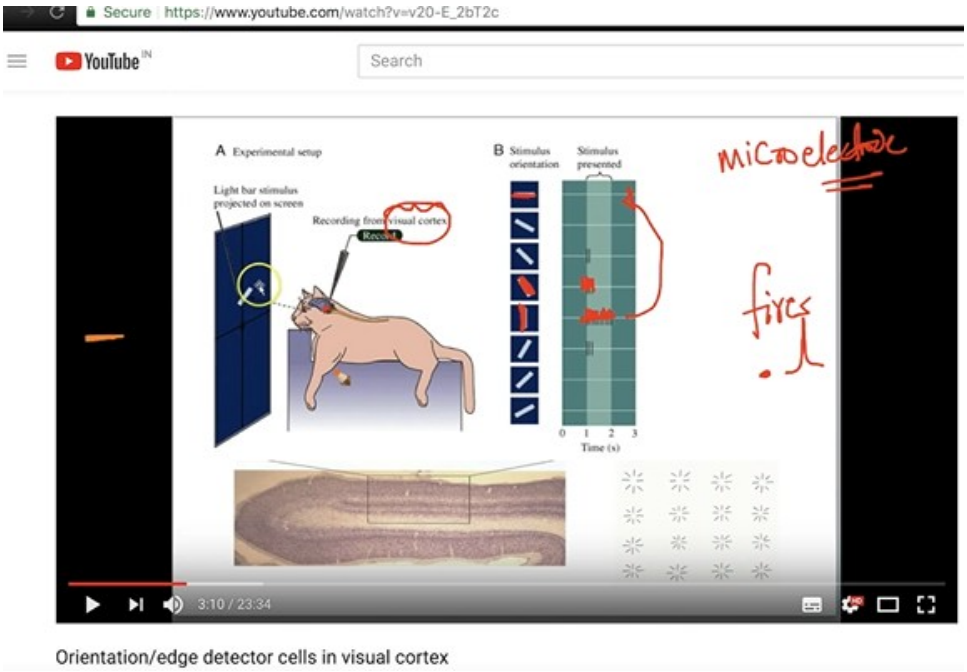
They are designed for the visual tasks, given an image to detect the objects in the image is called the object detection.



In 1981, there was the noble prize in medicine try to understand the human.

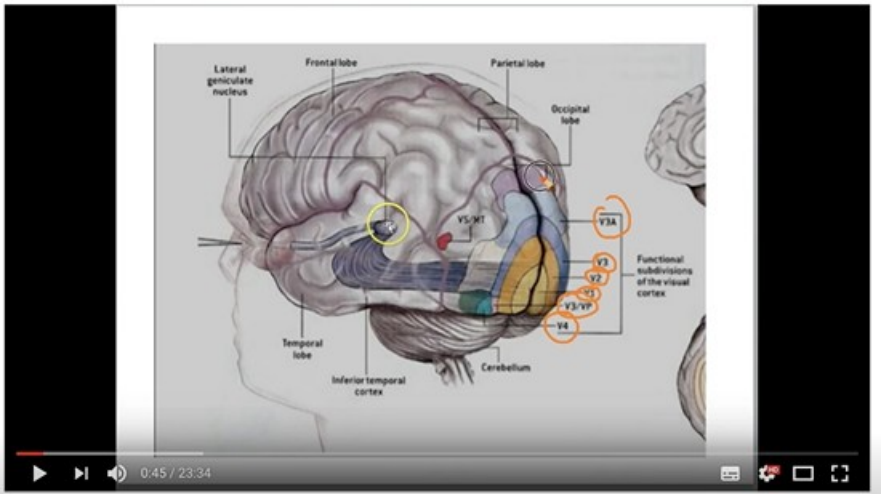
They show a light shown to the cat, the micro electrode is connected to the visual cortex and started recording. There are few neurons fire when sees the vertical and horizontal light.

Key finding of the research, there are some neurons in the visual cortex that fire when presented with lights of specific angles.

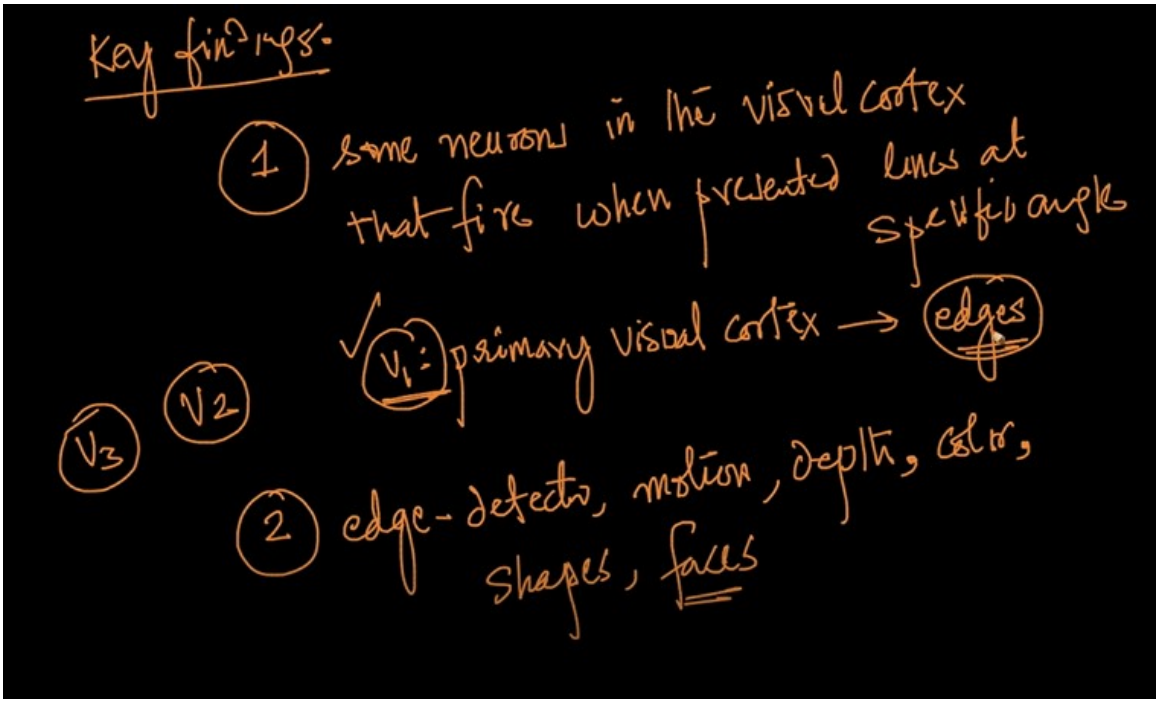


Orientation/edge detector cells in visual cortex

Visual cortex has multiple layers.



There is one area called v1 called primary visual cortex it detects the edges. Further it is concluded that the edge - detection, depth, color, shaped and faces.



There are more complex neurons for the complex tasks.

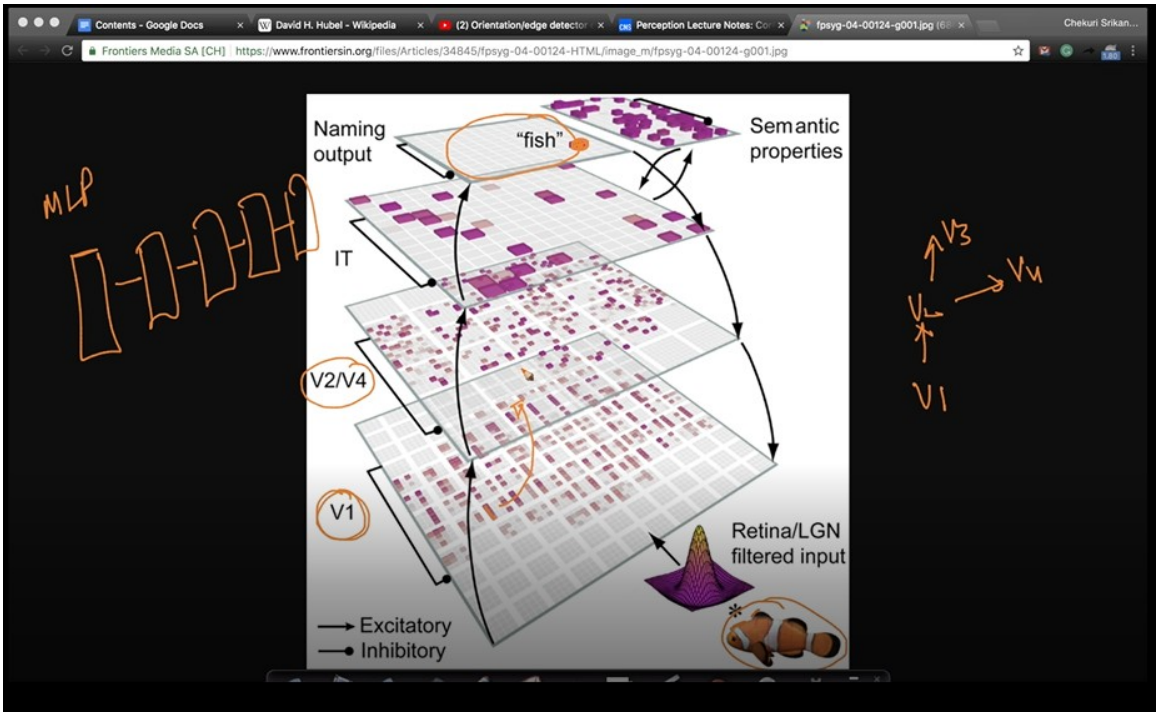
Functional specification:

Functional specialization

Match each visual area to its corresponding function:

V1	Motion
V2	Stereo
V3	Color
V3a	Texture segregation
V3b	Segmentation, grouping
V4	Recognition
V7	Face recognition
MT	Attention
MST	Working memory/mental imagery
etc	Etc.

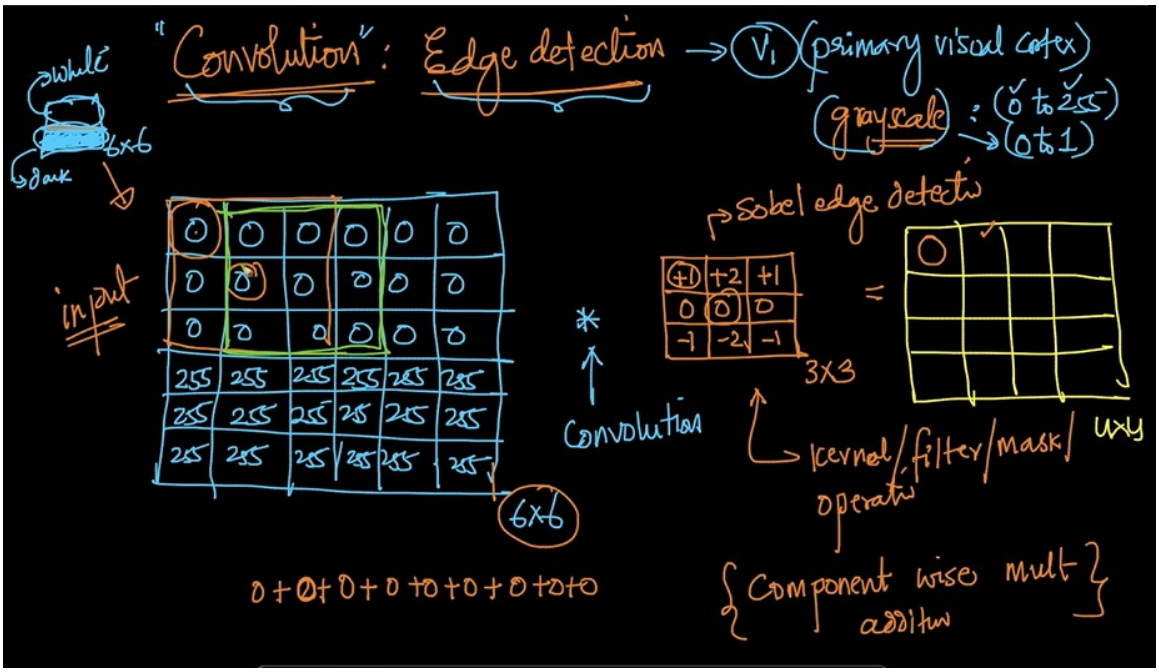
These all behave like layers, the lower layers looks at the edges and the higher layers have the more complex representations.



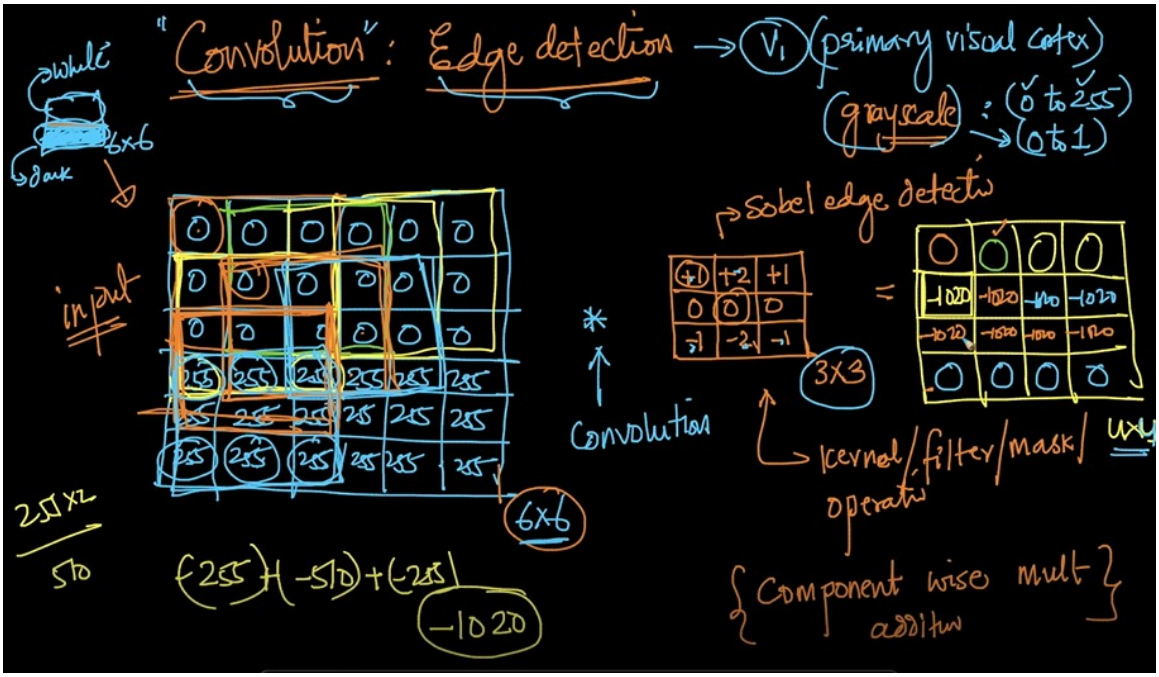
There is a nice hierarchical structure in the visual cortex.

Convolution edge detection: It is the primary visual sat.

Explanation:



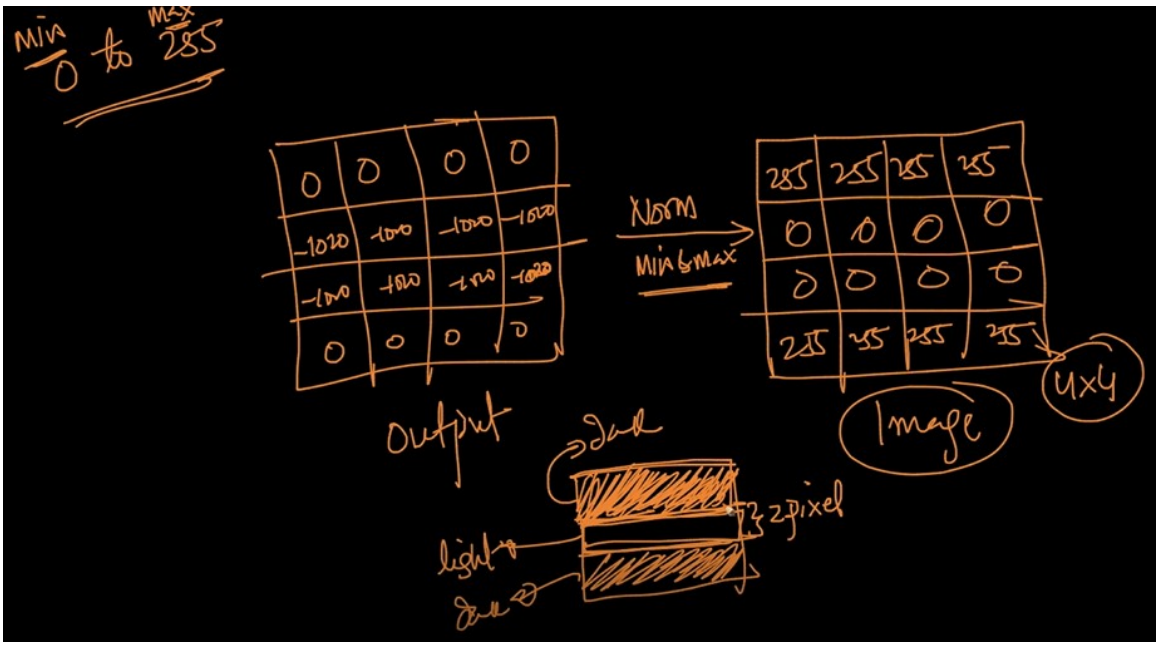
We will fill all the values in the yellow grid by moving one pixel upwards (or) downwards.



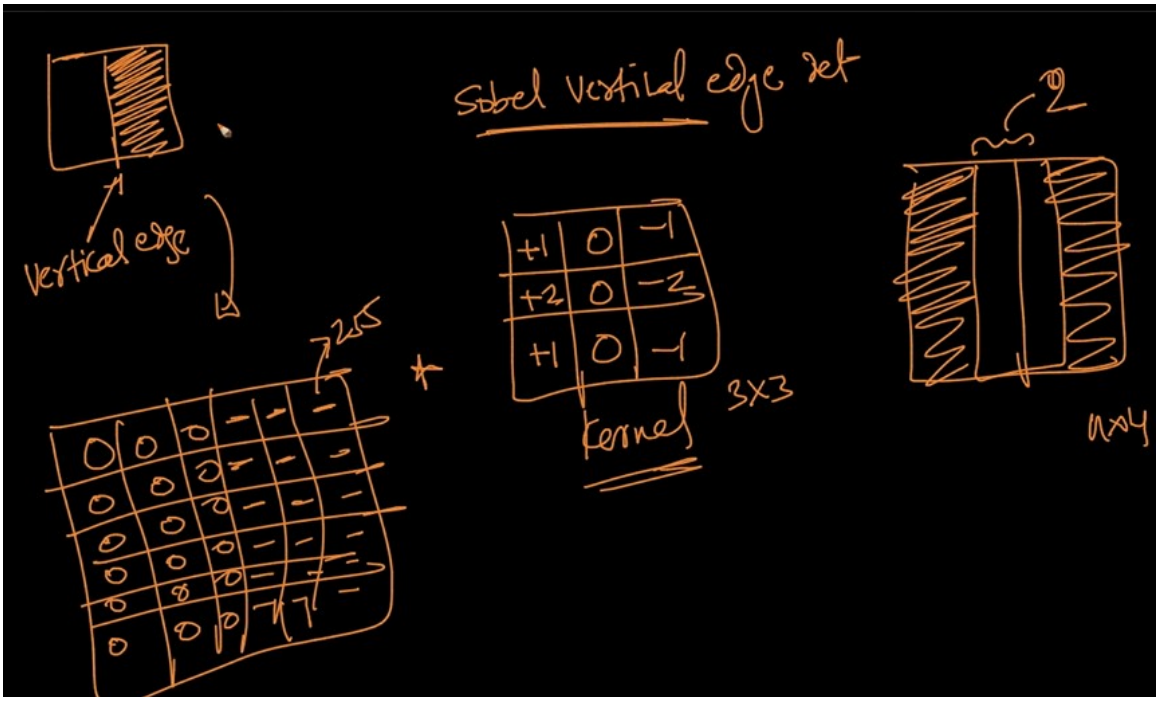
We cannot create more than 4*4 as the kernel is 3*3 on 6*6. This is the convolution operation.

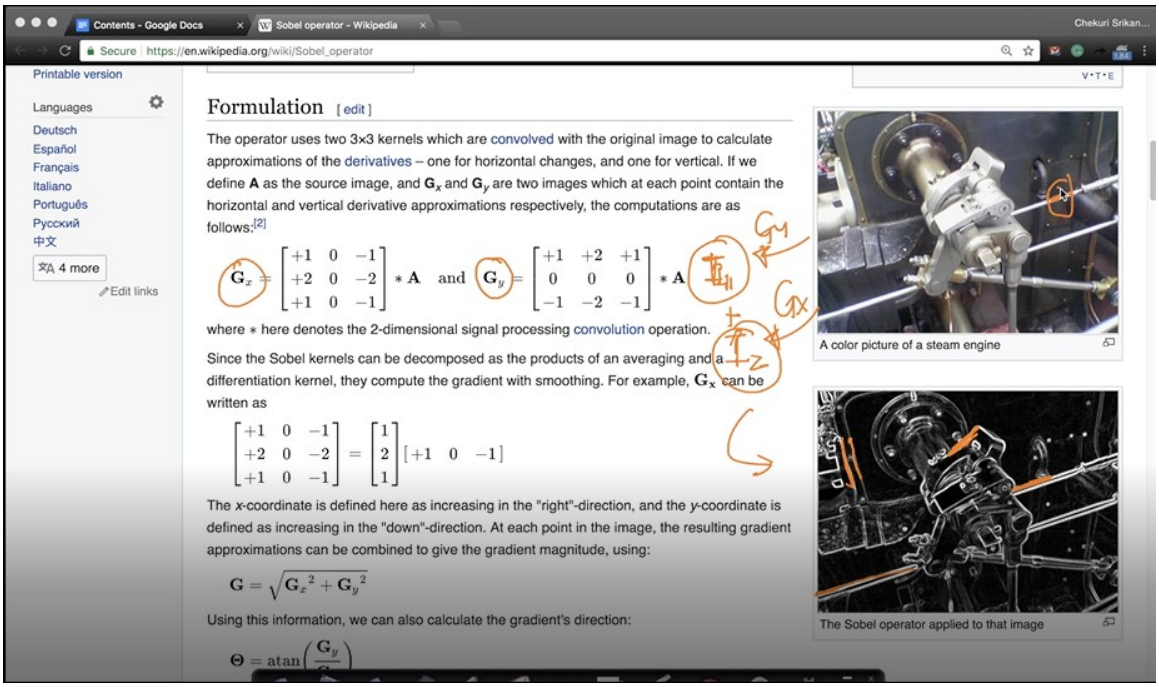
Describing the output image:

We will re normalize using the max-min on the output image. The output image is obtained as follows:



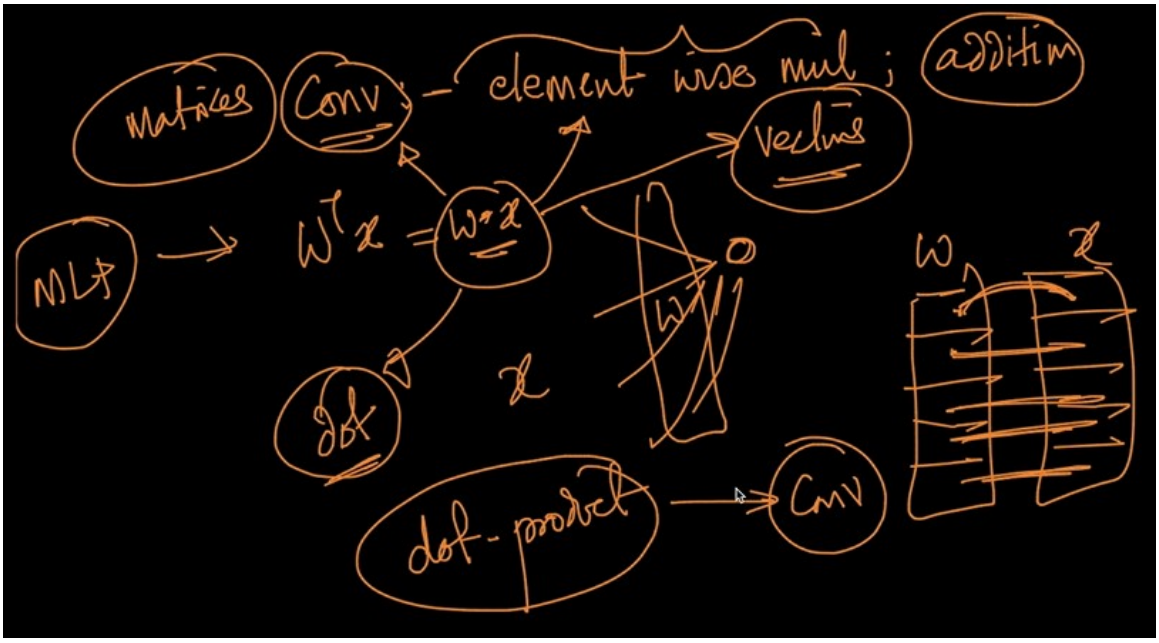
This is for horizontal edge, helps to detect the horizontal image. Here we have the vertical edge. then it is called the vertical edge detector.





In contrast we are using convolution operation to the image.

We are doing the element wise multiplication and addition. This is same as the MLP's.

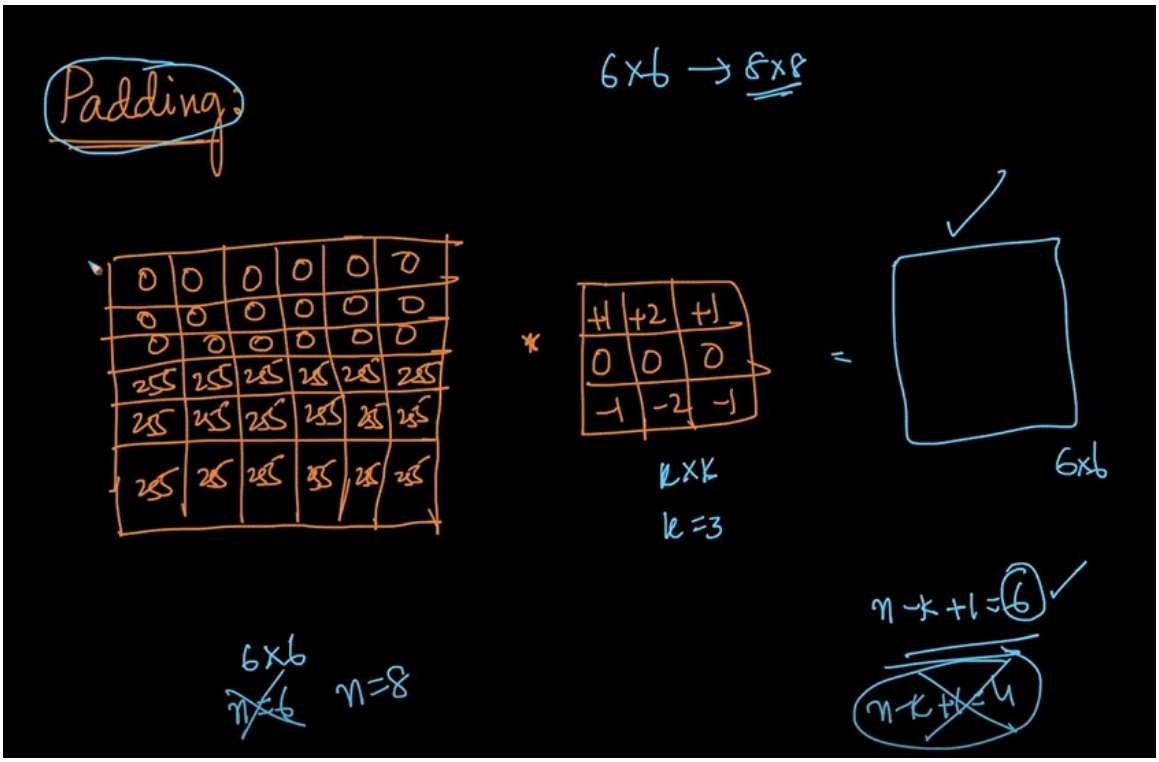


Padding and strides:

Padding:

The output that we want is same as input image dimensions, we will do padding to achieve this.

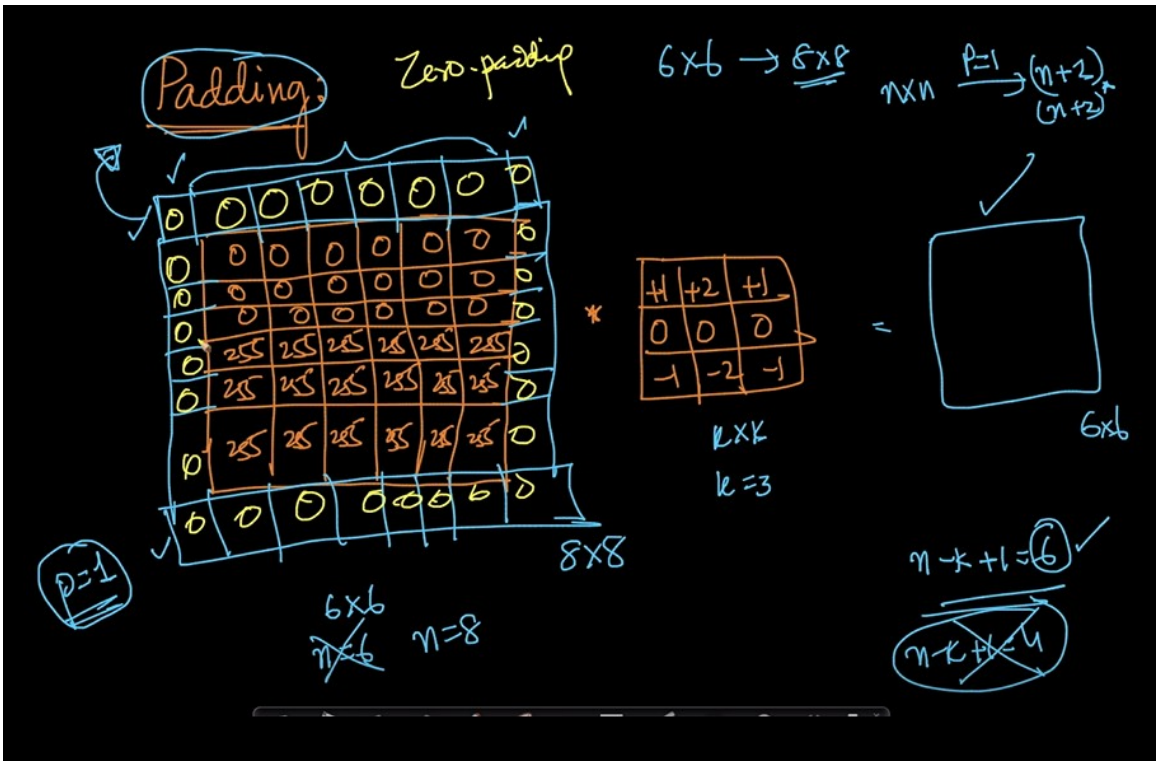
Before padding:



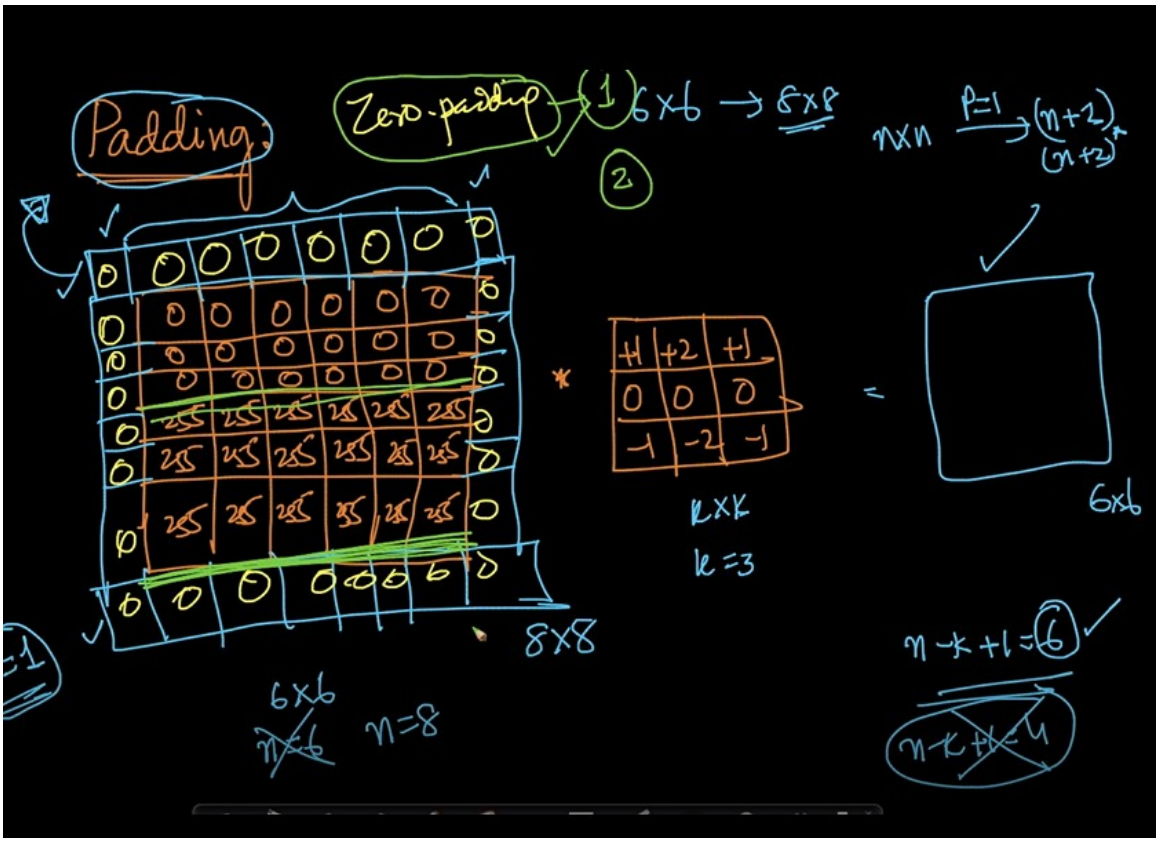
After padding:

Technique 1:

The simplest way is zero padding.

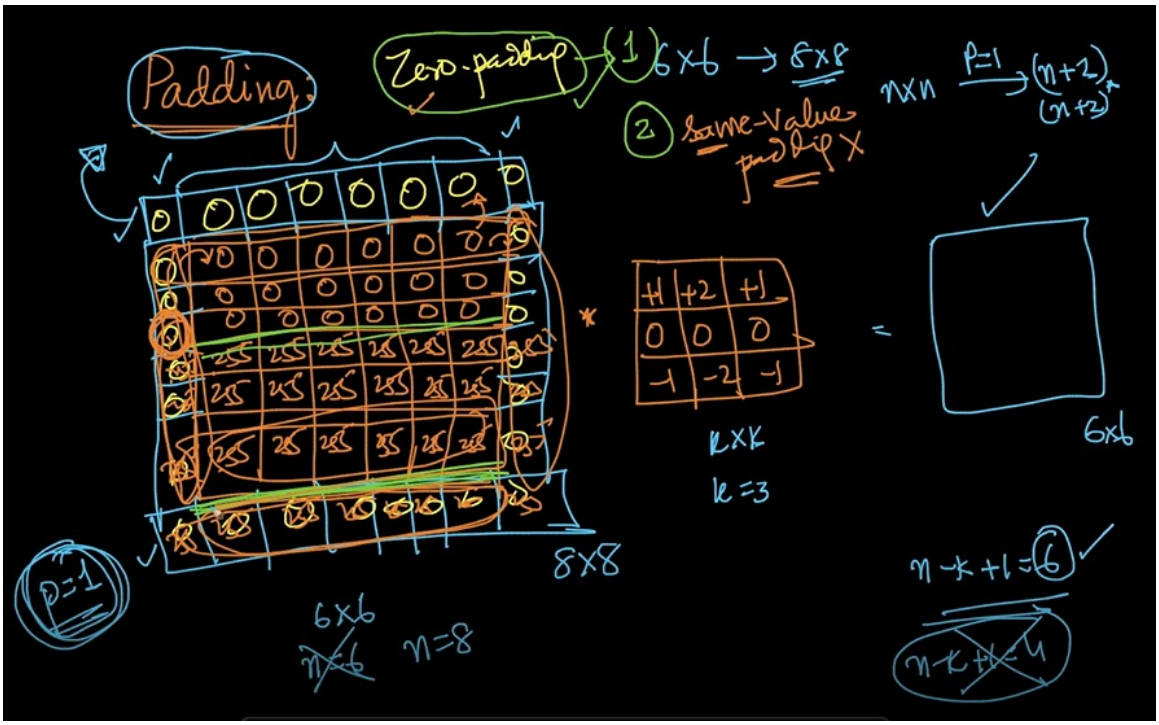


Initially we have one edge, by zero padding we make extra edges.



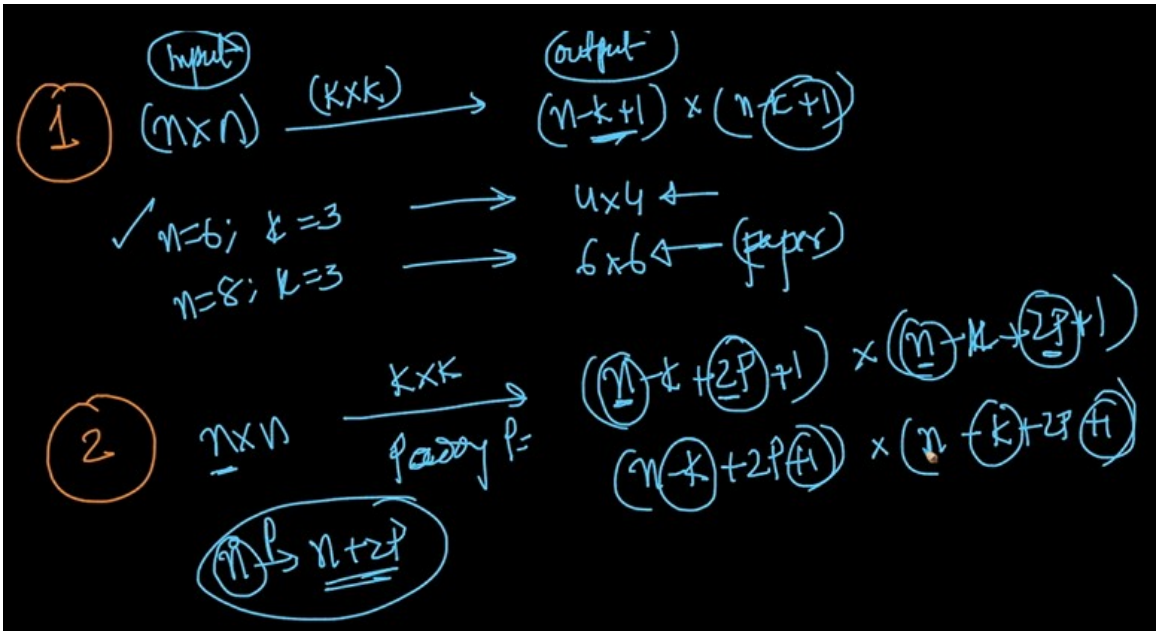
Technique 2:

Instead of zero, we can also pad with the nearest value. This is called the same value padding. We will take the values of the column next to it. **IT IS NOT USED EXTENSIVELY IN WHOLE CONVOLUTIONAL NEURAL NETWORKS.**



They are typically square matrices.

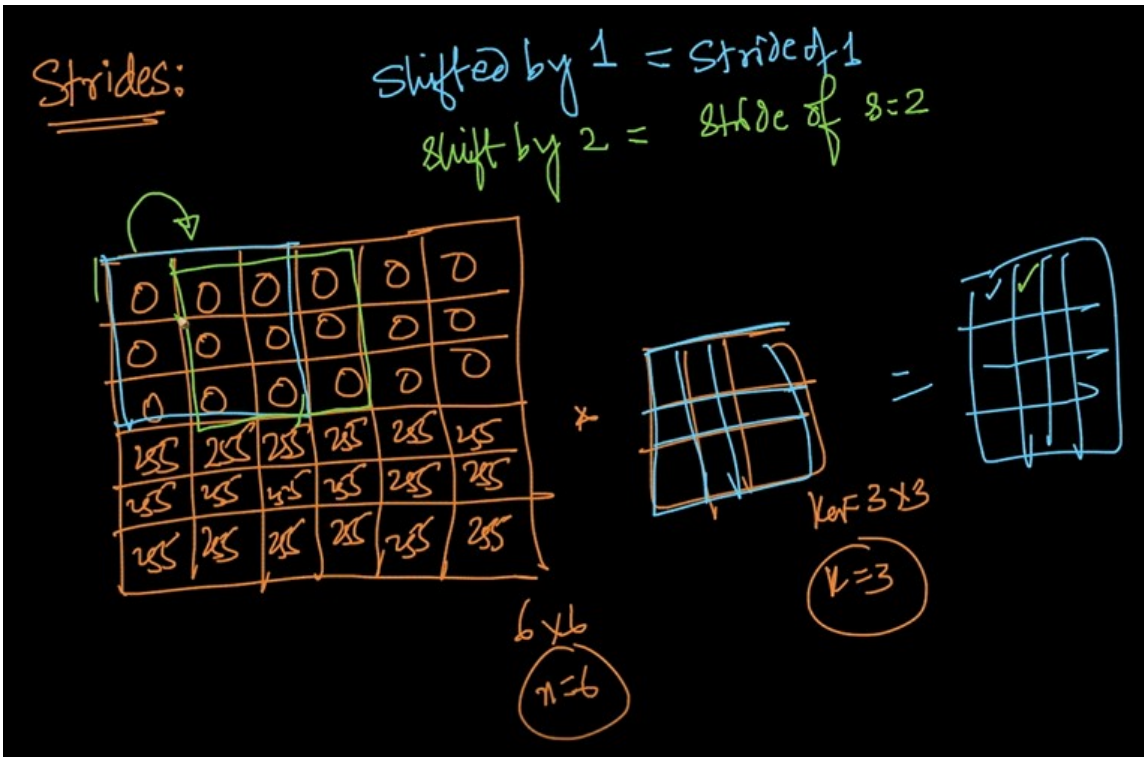
Formulation:



Strides:

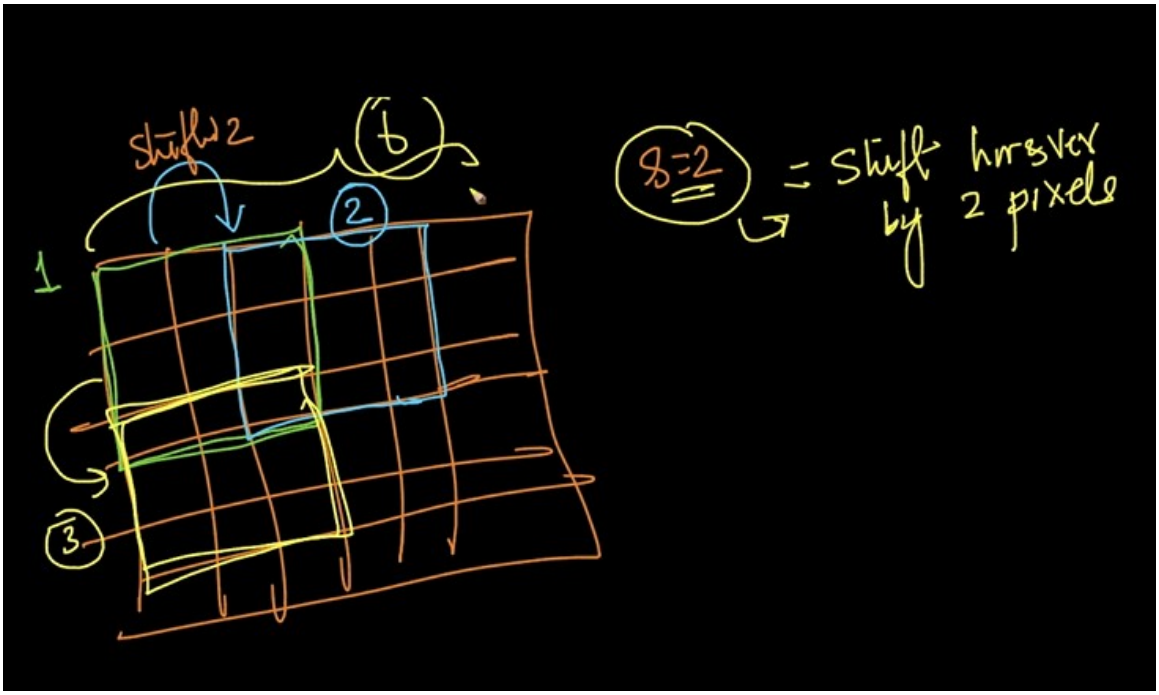
Stride is by how much the kernel is shifted on the image, we can shift by 2 this is called two stride convolution operation.

Stride of 1:



Stride of 2:

It means we are shifting horizontally and vertically by 2 pixels.

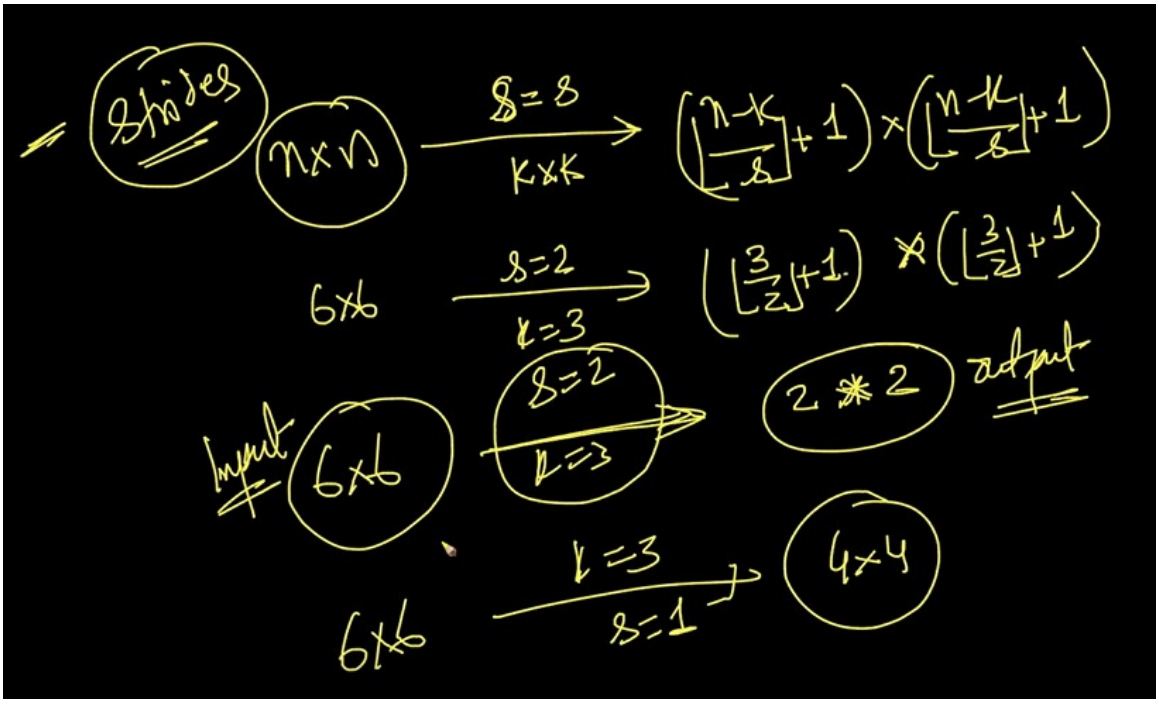


Stride helps in reducing the size of the matrix. Here we are performing a division operation.

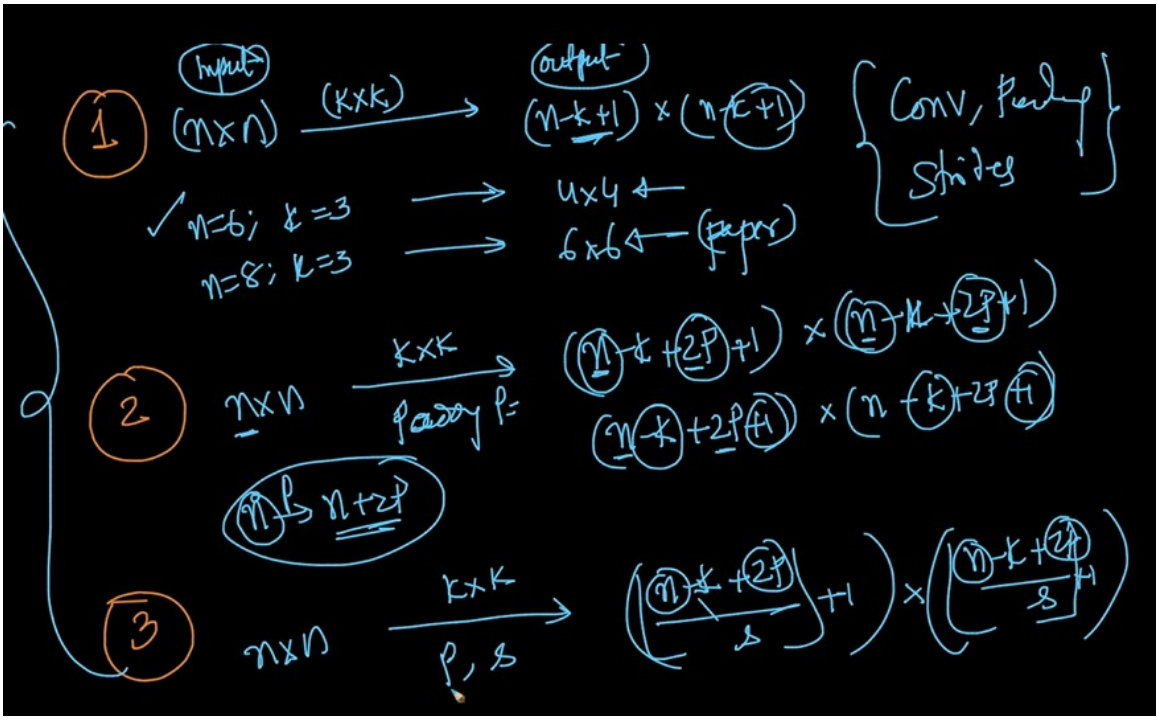
We will take the floor of the division operation.

Formulation:

This gives the significantly smaller matrices.

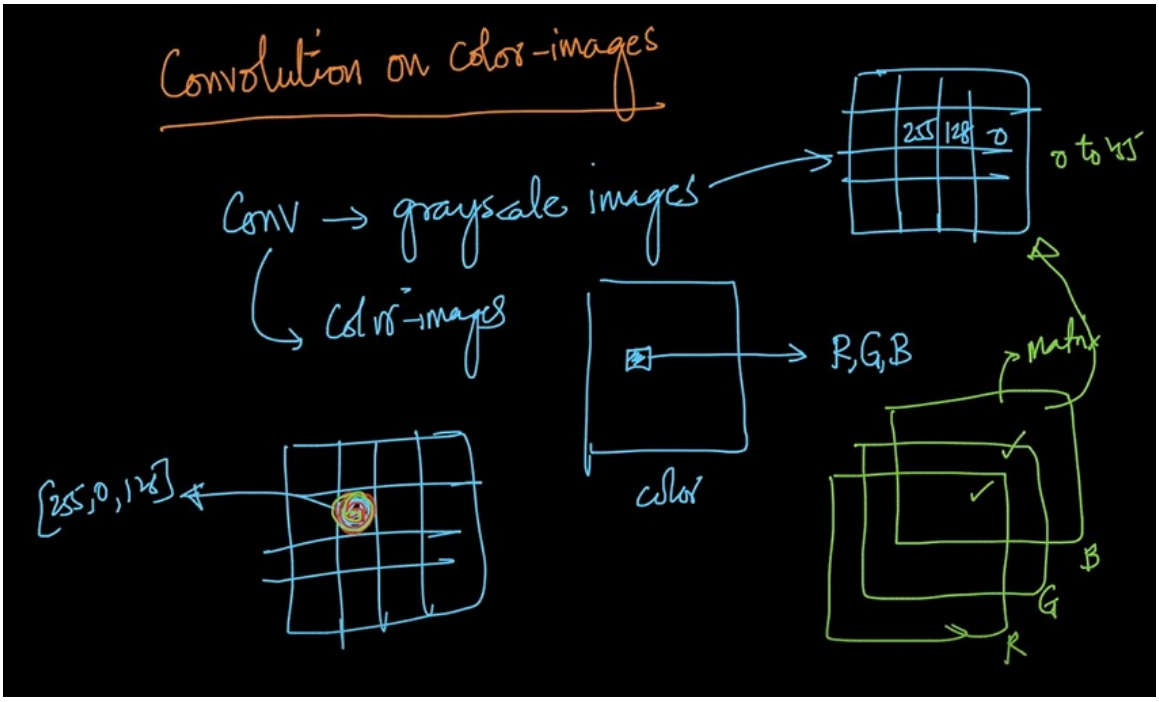


Formulation of padding and strides:

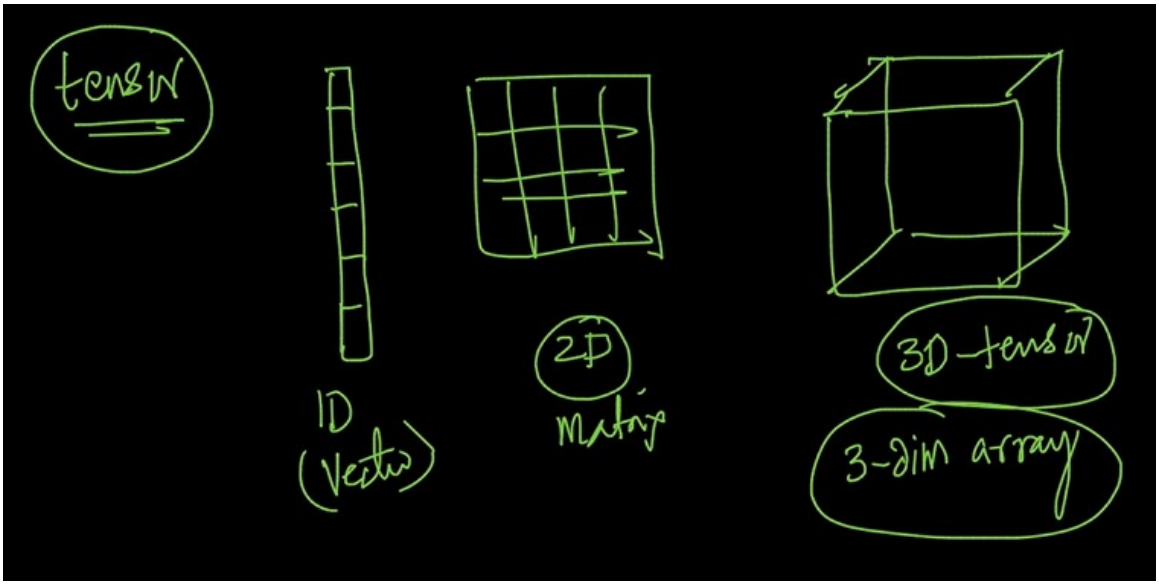


Convolution on Color images:

we have the three colors red, green and blue in an image.

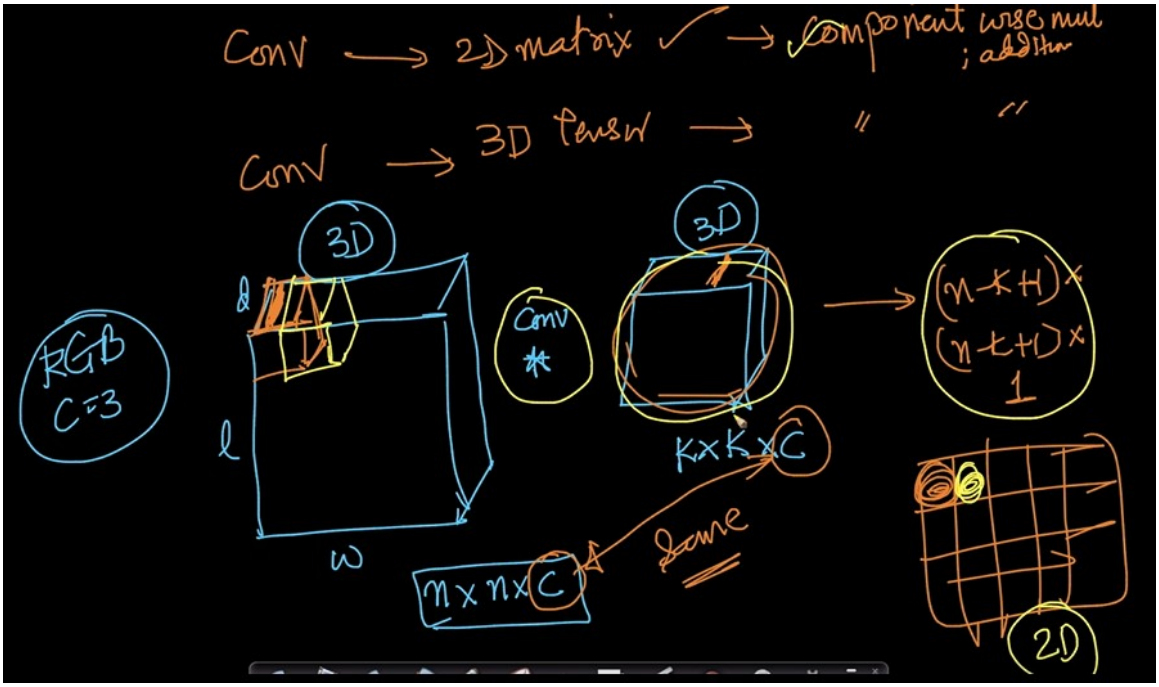


Tensors:

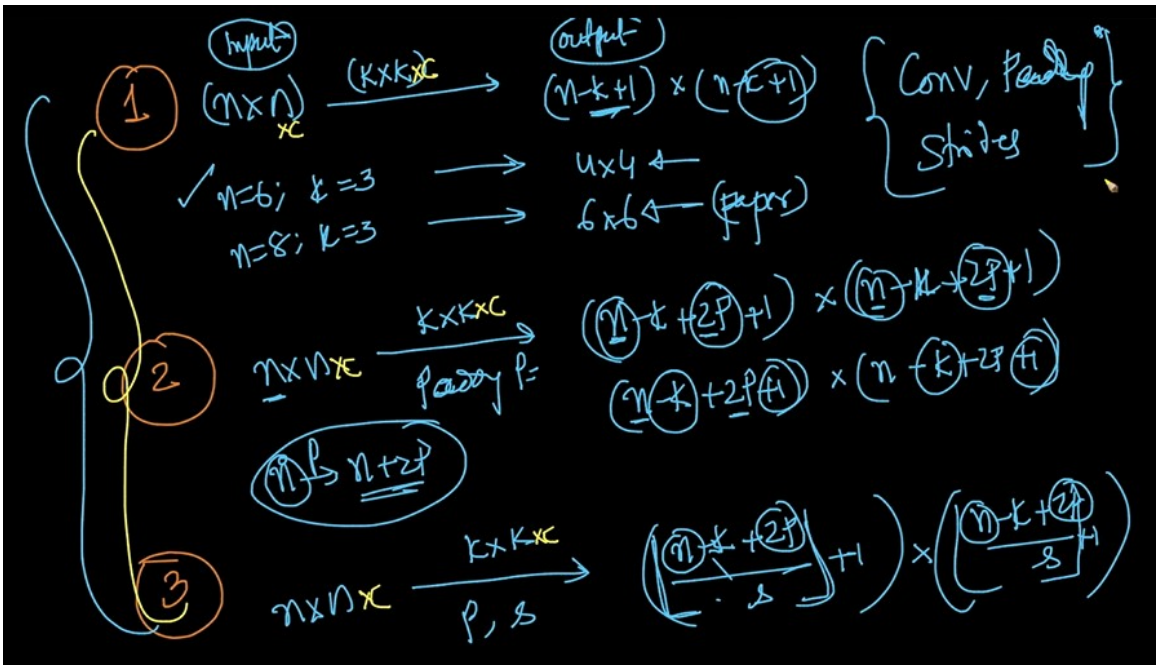


A color image can be represented as the 3d image, we refer to as channels.

If we have the 3d tensor, then we have the 3d kernel with the same channels to give the one output value.



Formulation of the tensor processing in the convolutional neural networks.

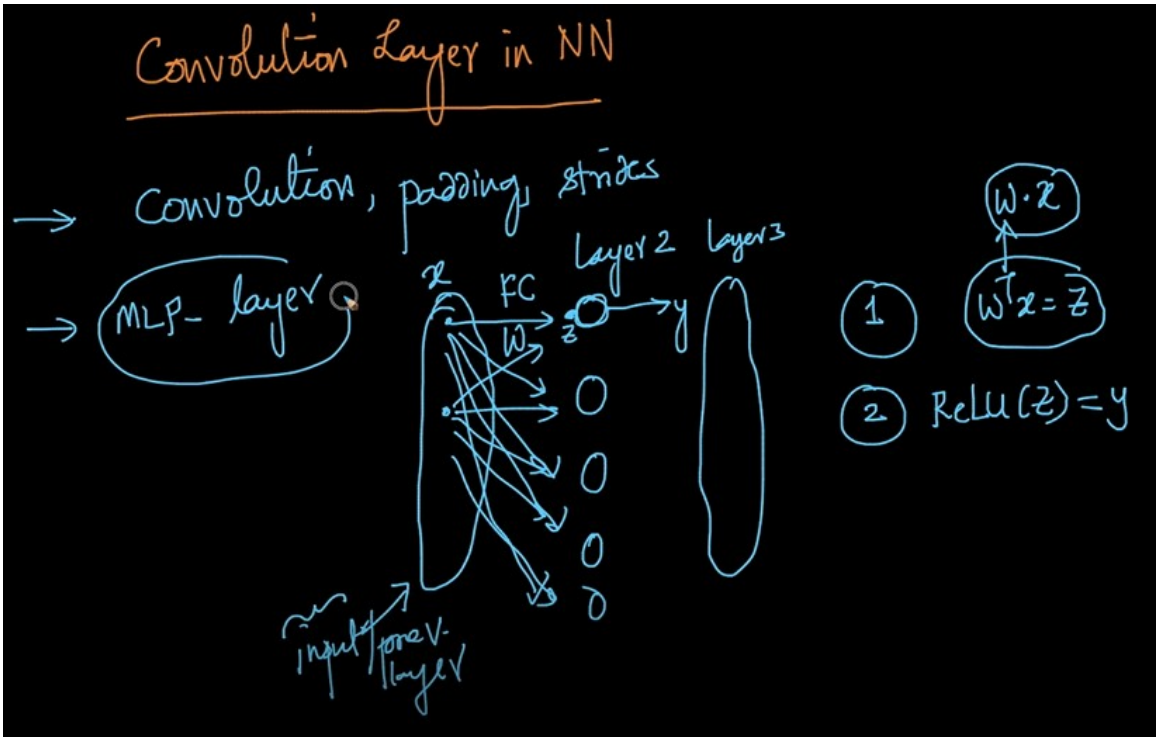


Except the channel parameter everything is same.

Convolutional layer:

Lets define one layer of the convolutional layer.

Analogy of MLP:

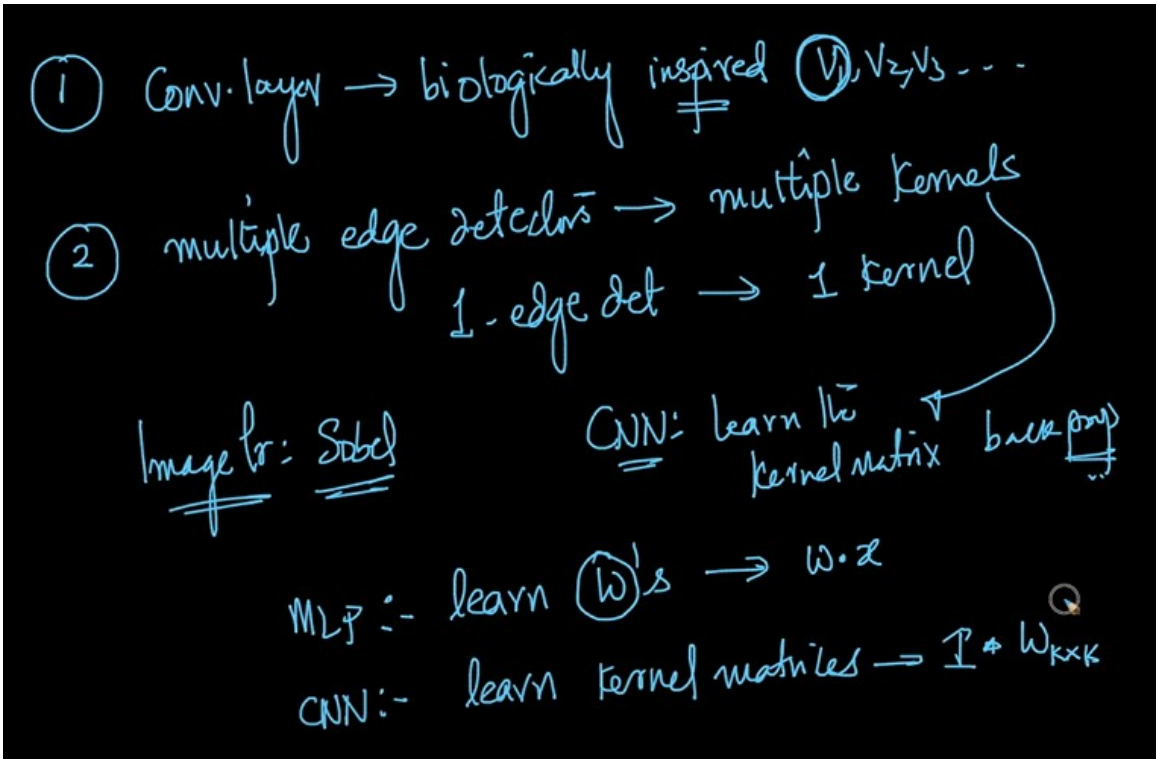


Convolutional layer:

We are designing the conv layer as the biological inspiration of V1, V2, V3 of the brain.

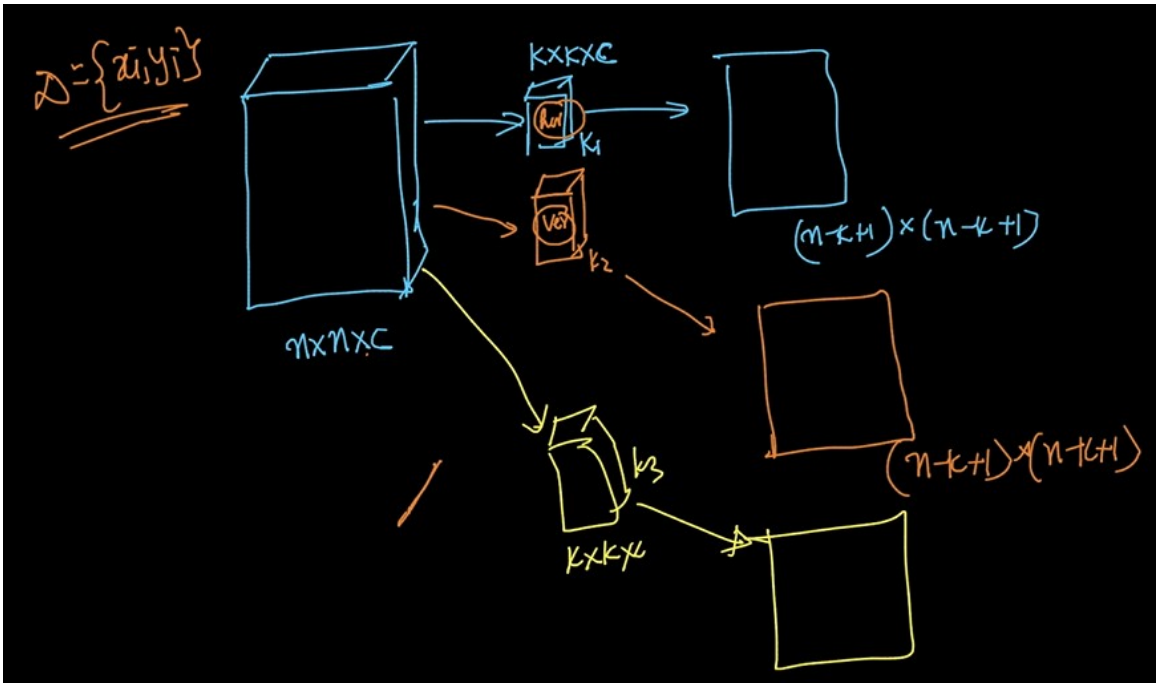
In CNN's we learn the kernel matrix using the backpropogation algorithm.

Kernel matrices are the weight matrices in CNN.



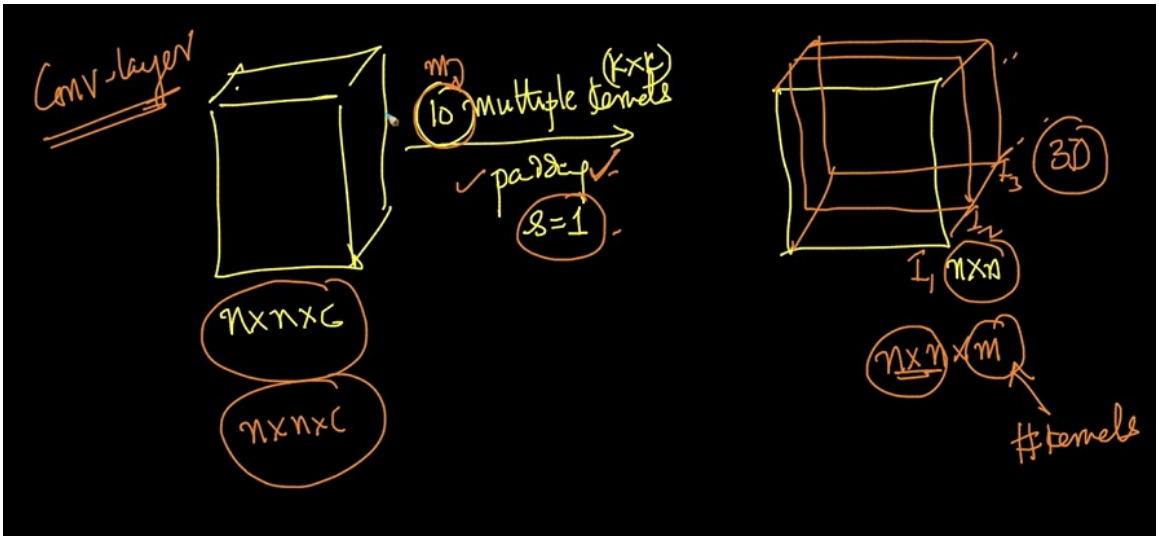
The way we design one layer in conv layer is as follows:

Instead of one kernels we use multiple kernels which act as the weights in the CNN's. We can have the different kernels that detect the various angles and positions in the network.



If we have the 10 kernels we get the 10 output images after performing the convolution operation.

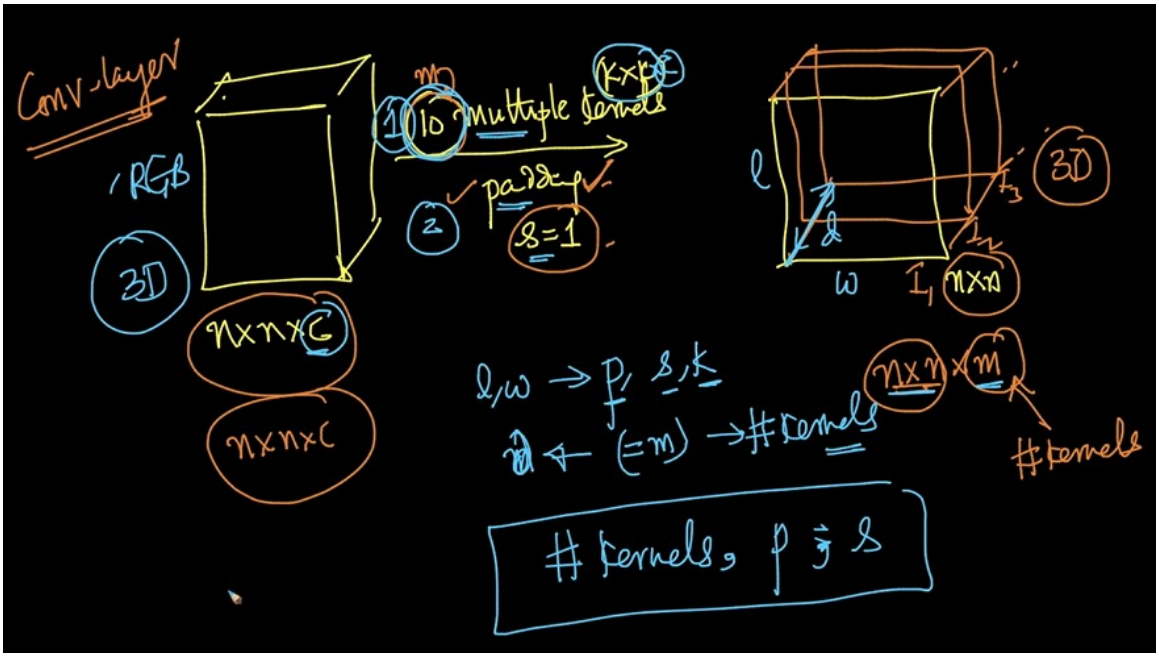
The shape of the output image is dependent of the stride and padding of the kernels on top of the original image.



The length and width is dependent on the padding, strides and kernel dimensions.

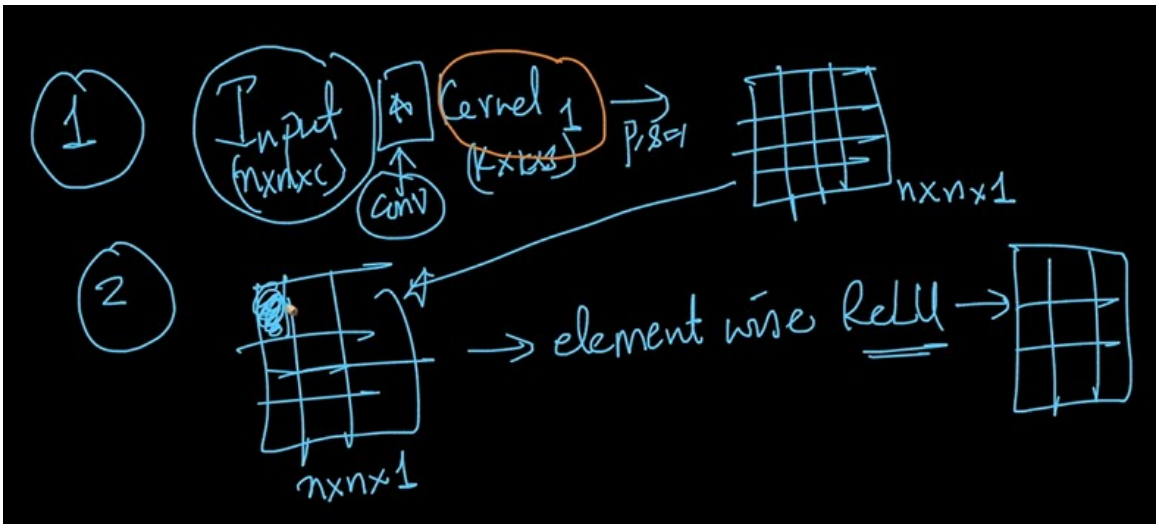
The number of layers of images that are outputted is dependent on the number of kernels which is hyper parameter.

Hyper parameters in CNN's:

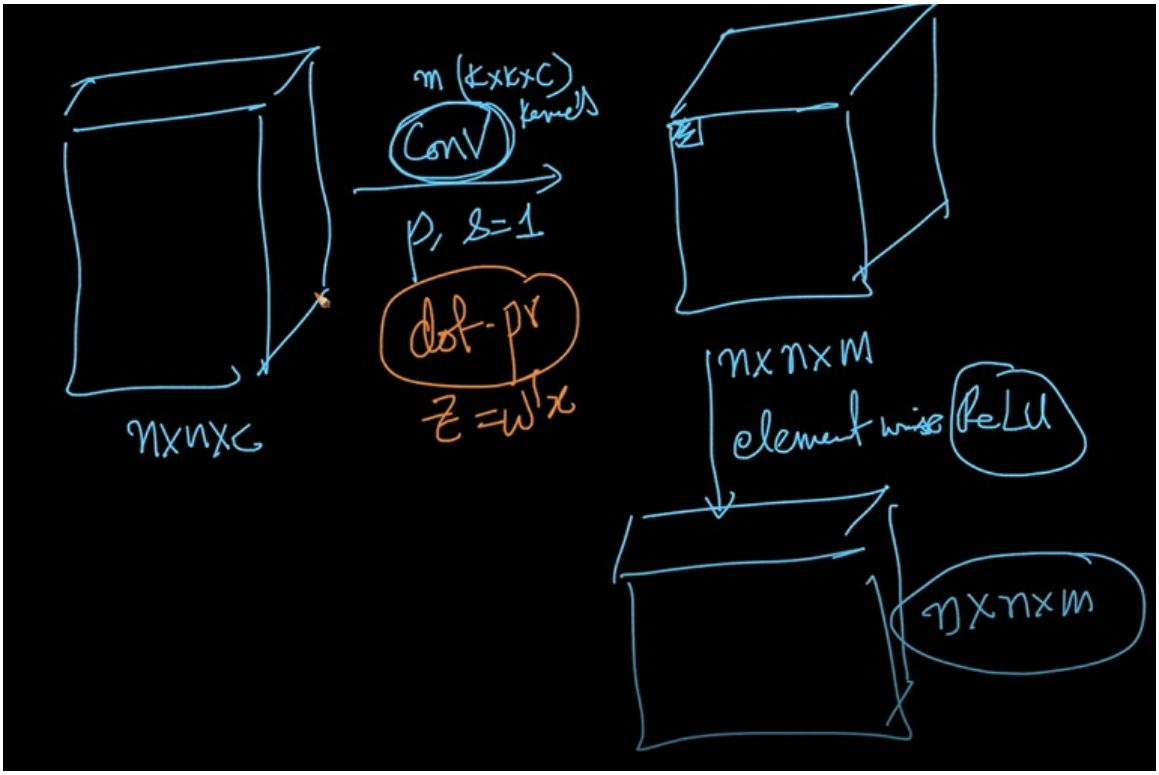


In the first step, take the input image and convolve with the kernel 1 and we get the output image.

after this we apply element wise ReLU operation.



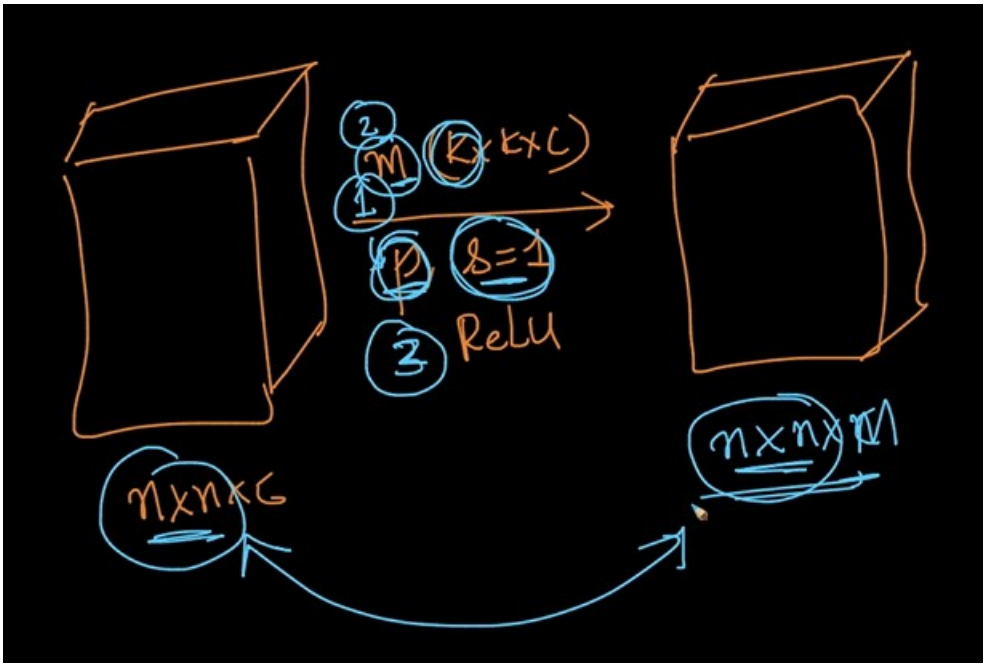
An actual convolutional layer looks like this,



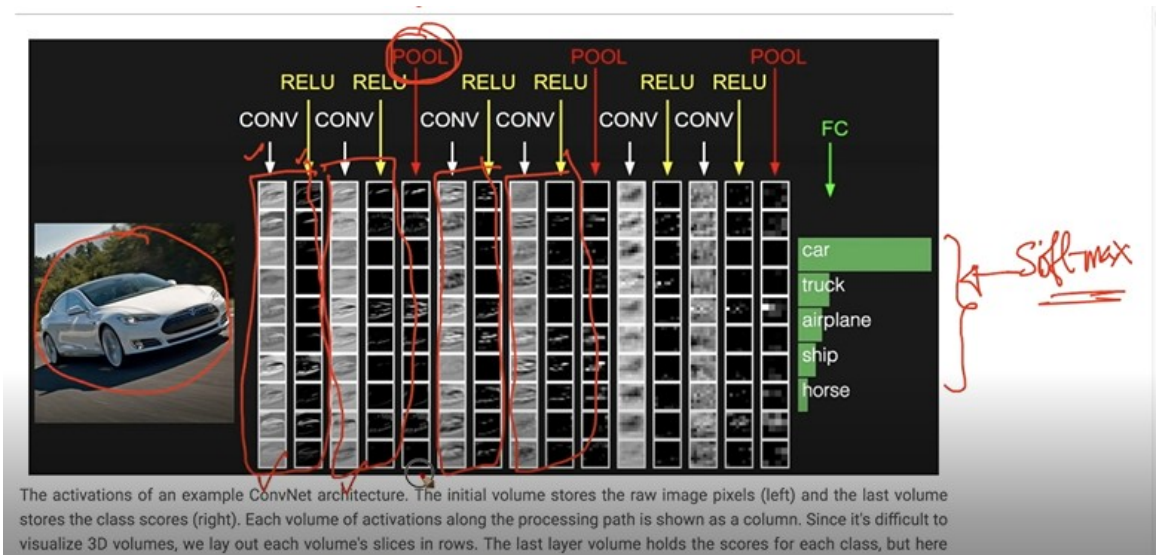
That is the relationship between the MLP's and CONV operations. It is often shown as one step.

The output dimensions need not to be the same as input. Because it is dependent on padding, stride and kernel dimensions.

There will be 'm' output images, where 'm' is the number of kernels.

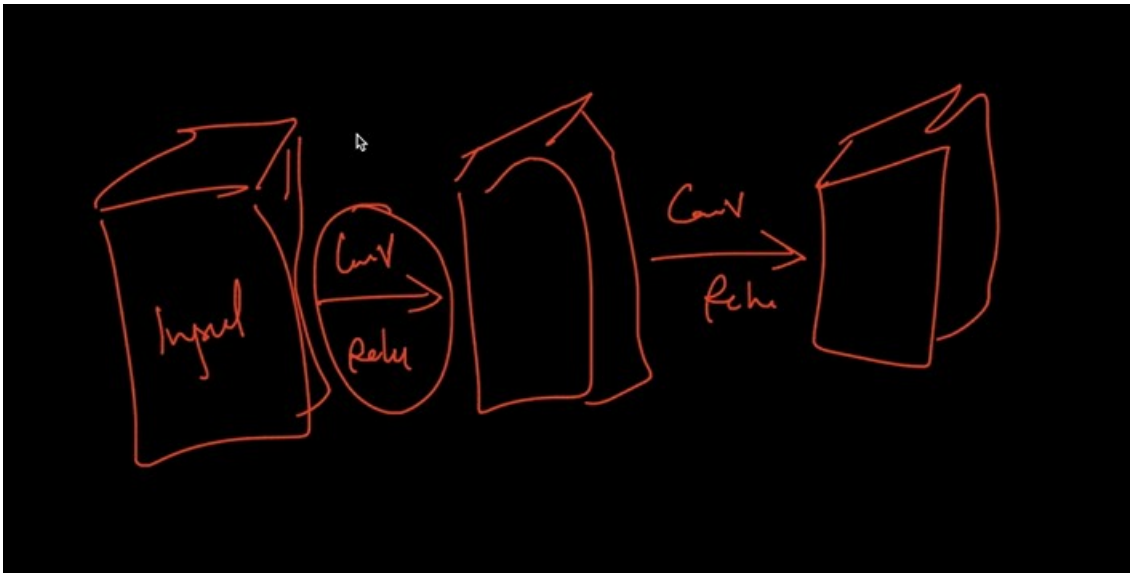


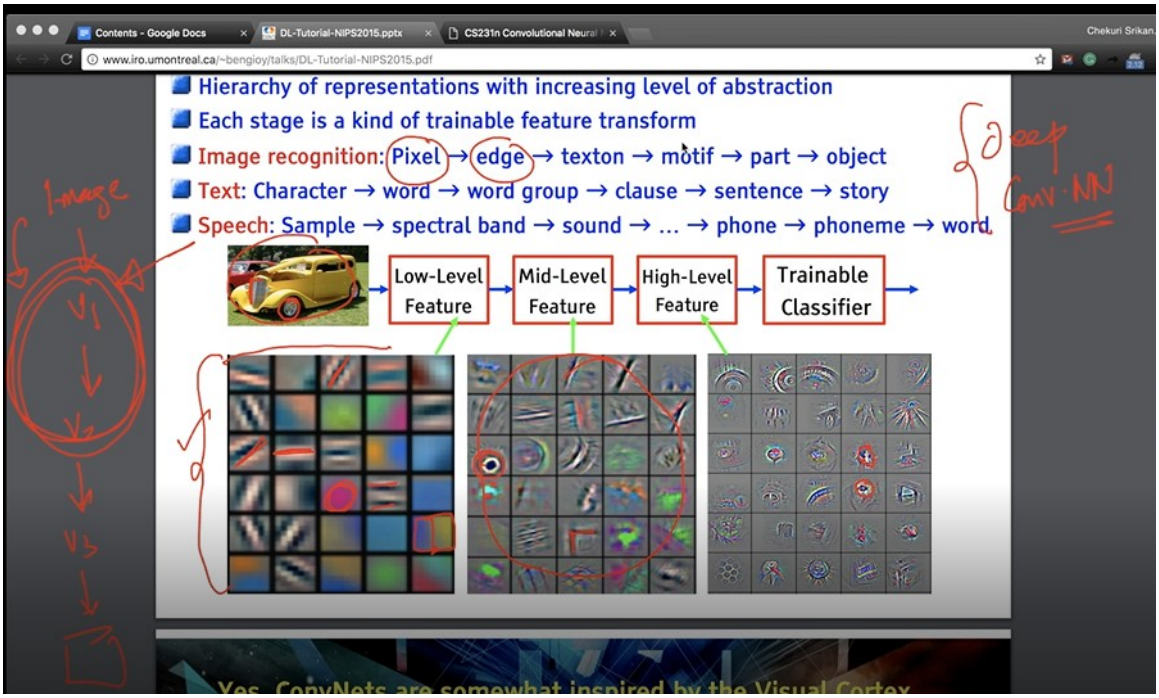
CONV Layer = CONV with multiple kernels + Activation units.



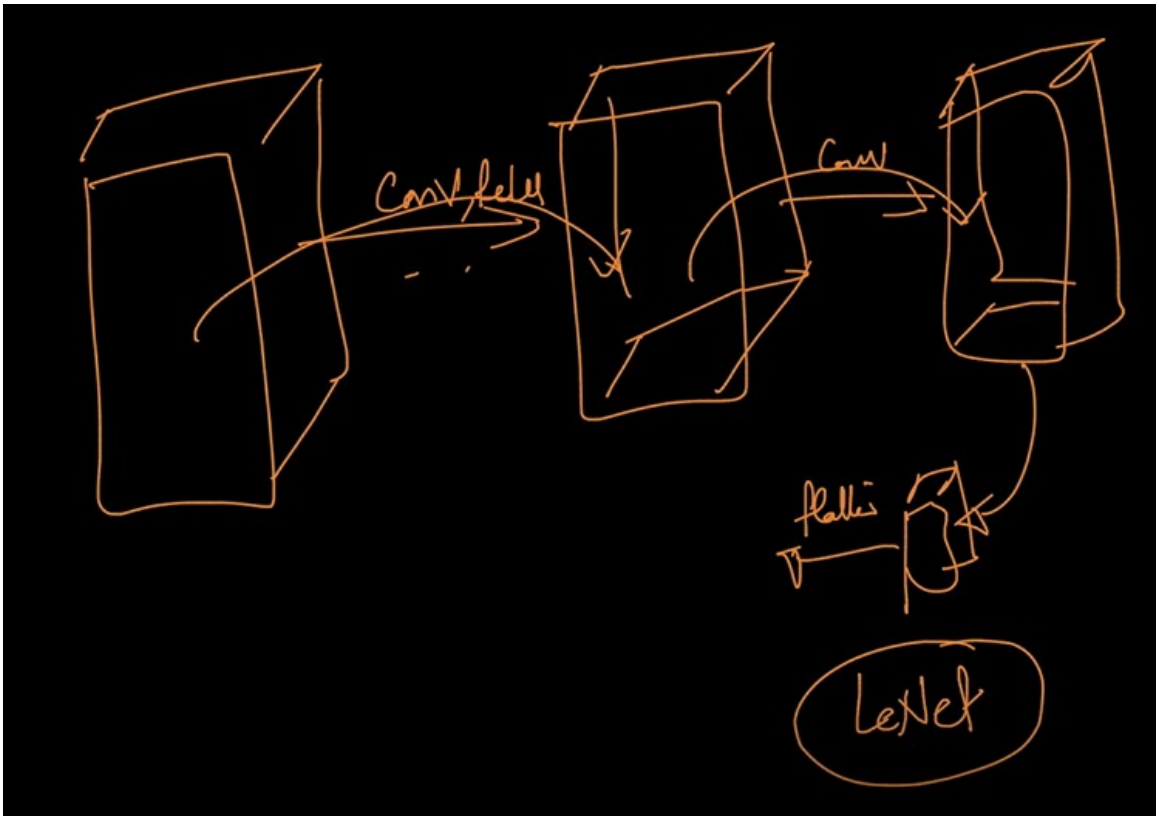
The activations of an example ConvNet architecture. The initial volume stores the raw image pixels (left) and the last volume stores the class scores (right). Each volume of activations along the processing path is shown as a column. Since it's difficult to visualize 3D volumes, we lay out each volume's slices in rows. The last layer volume holds the scores for each class, but here

This is how a real CONV layer looks like.





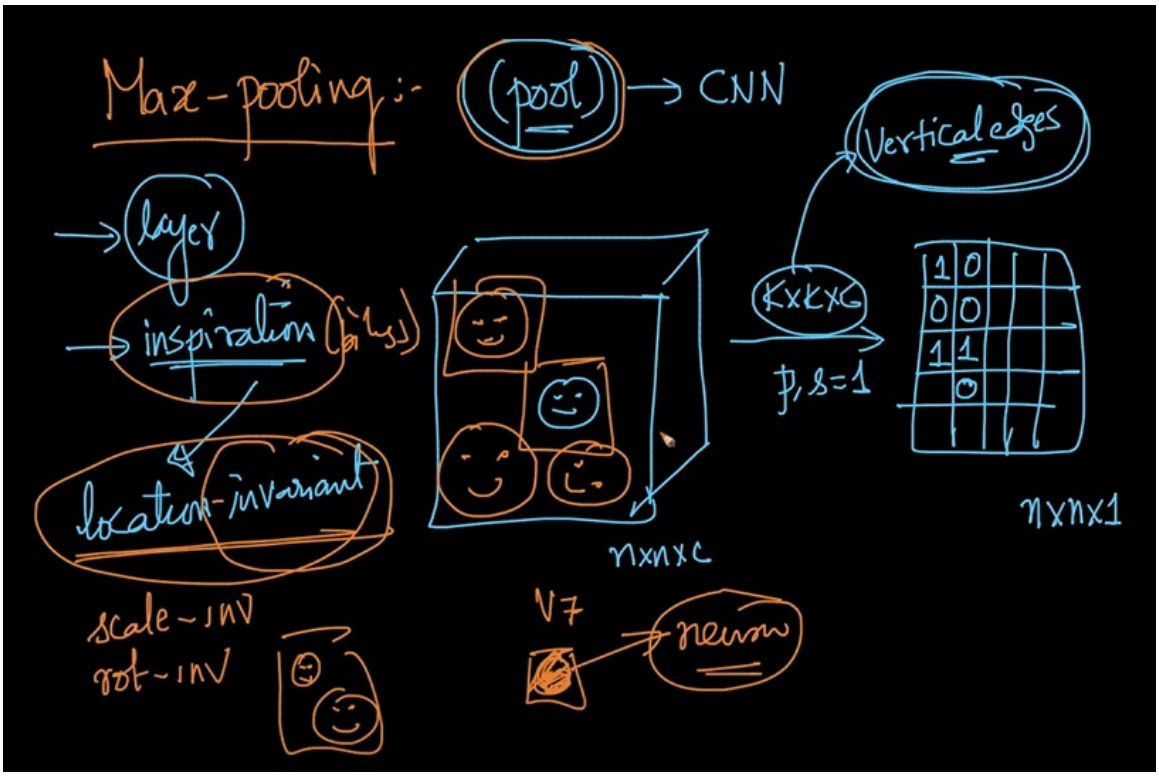
If we train a deep multi layer ConvNet, behaves very similar to how the visual cortex is working.



Max - pooling:

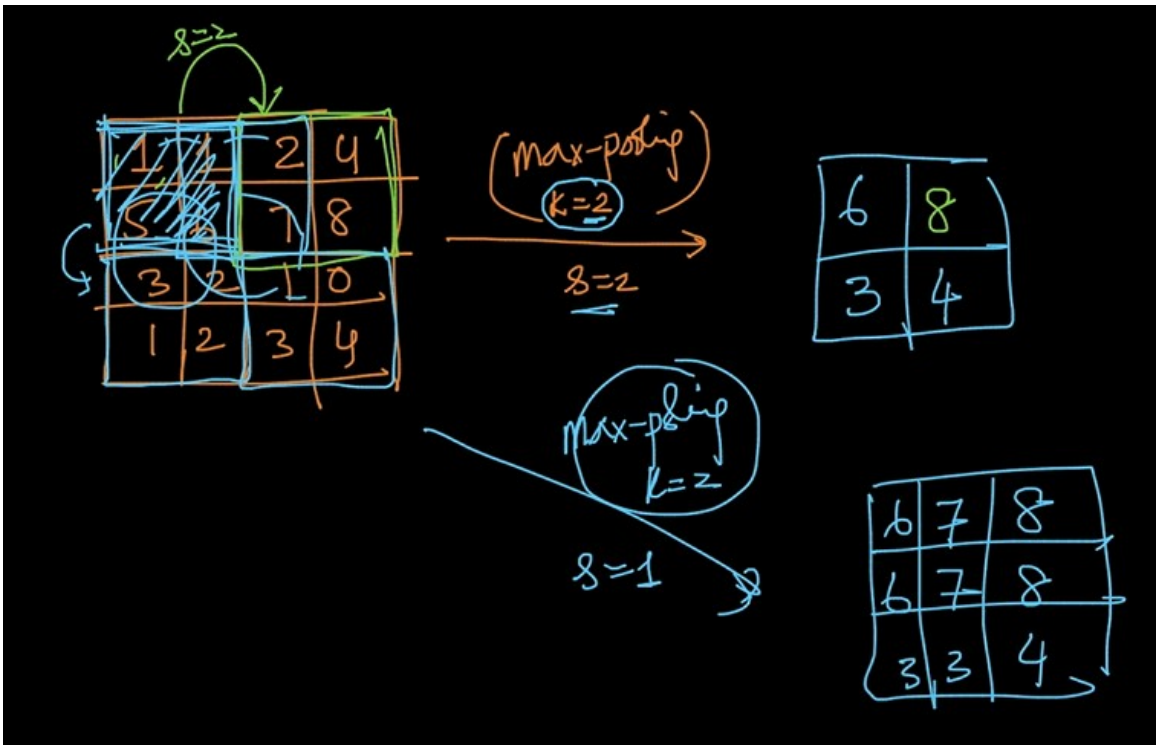
This pool is like a layer that can be added on the learned kernel. We want the models to be the location invariant, scale invariant and rotation invariant.

Pooling makes the network slightly invariant to these in-variants.

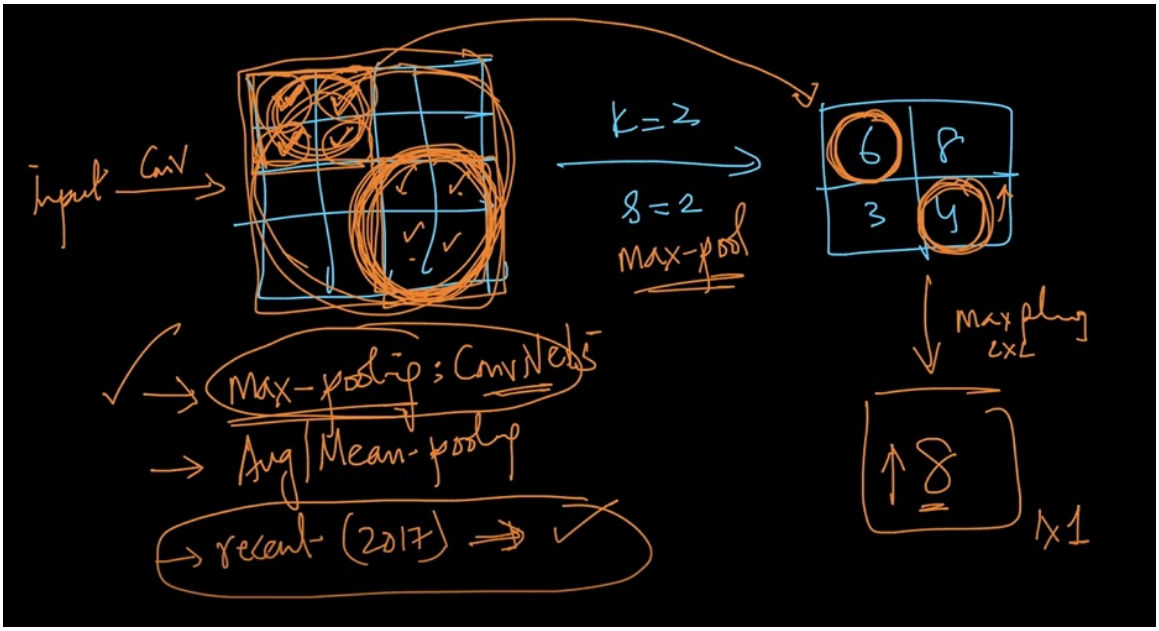


We can replicate this in-variance using pooling,

We do max pooling with kernel of size 2, It also has stride value.



How does max – pooling create in variance?

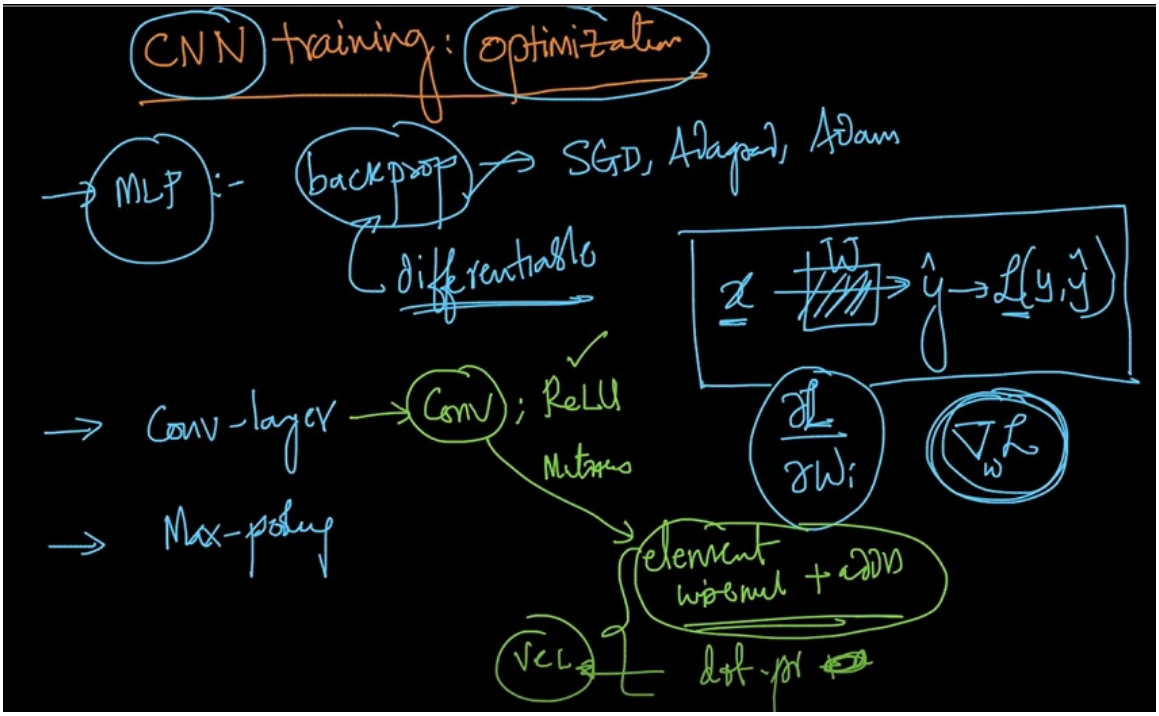


It is actually creating location in-variance. There is average pooling, mean pooling also. But Max pooling is most popular.

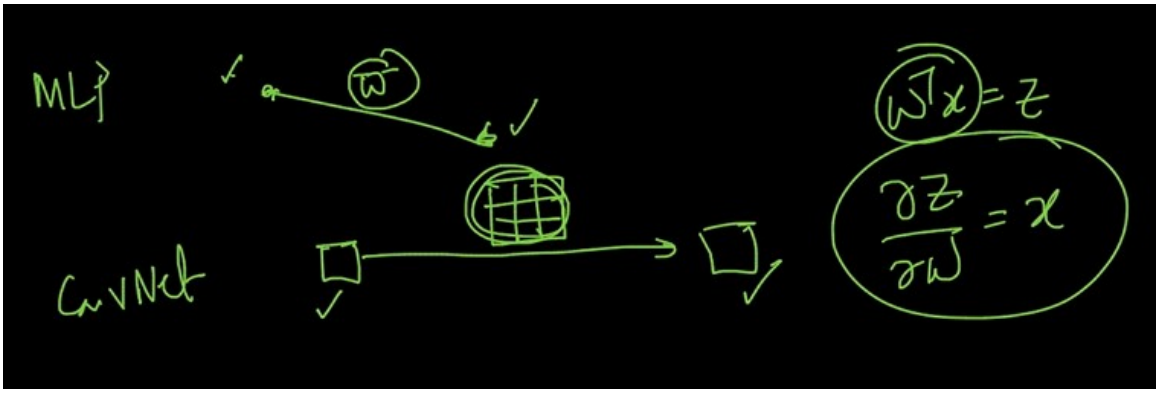
Conv, ReLU, Max Pooling are most commonly used operations.

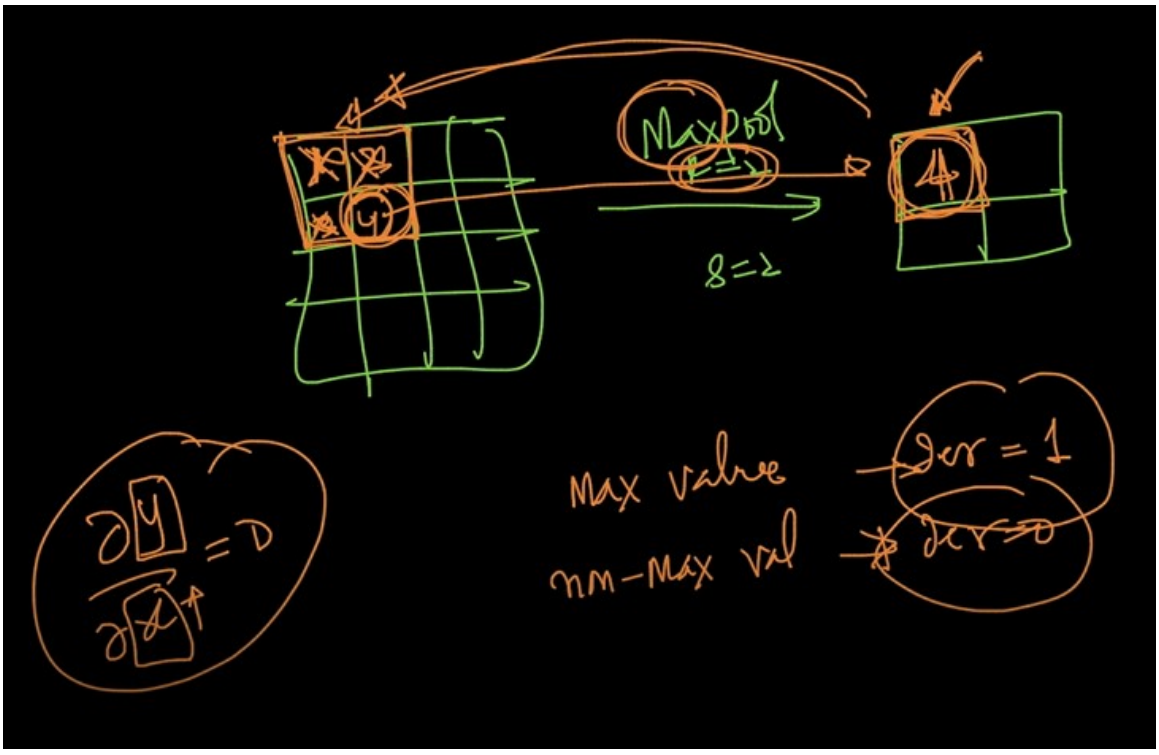
CNN Training optimization:

As long as we can compute the derivative, we can do back propagation using SGD, Adagrad and Adam.



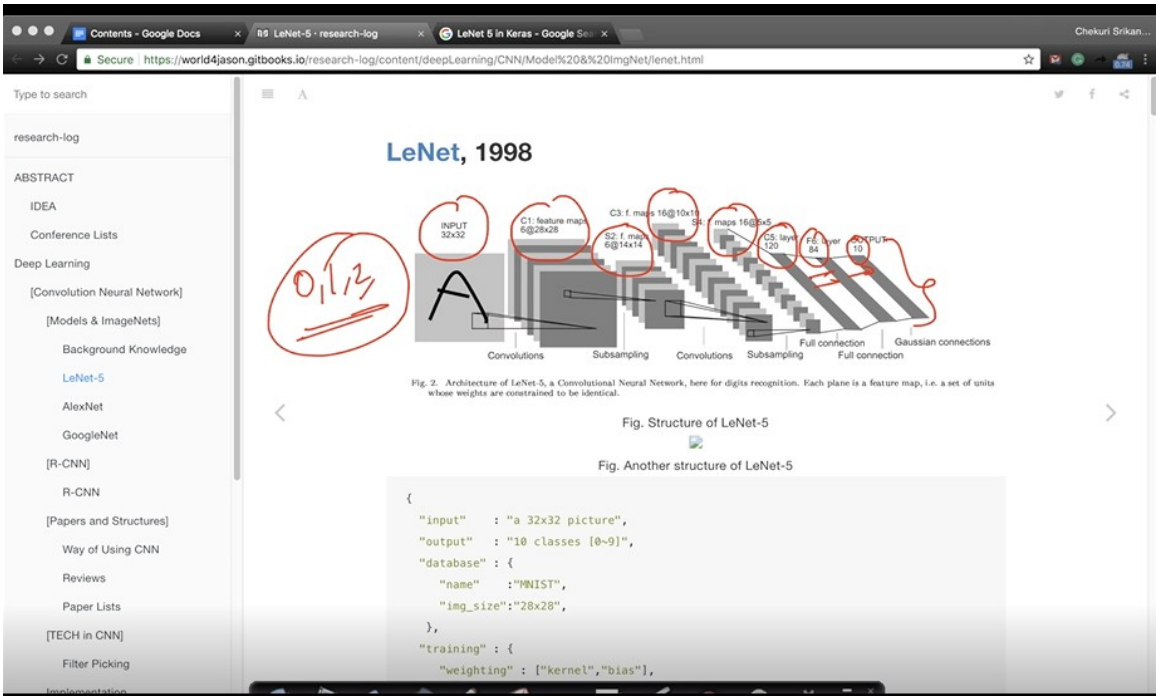
In conv net we do matrix product, this is also differentiable.





We give 1 for max value, and zero for the rest of the values.

LeNet:



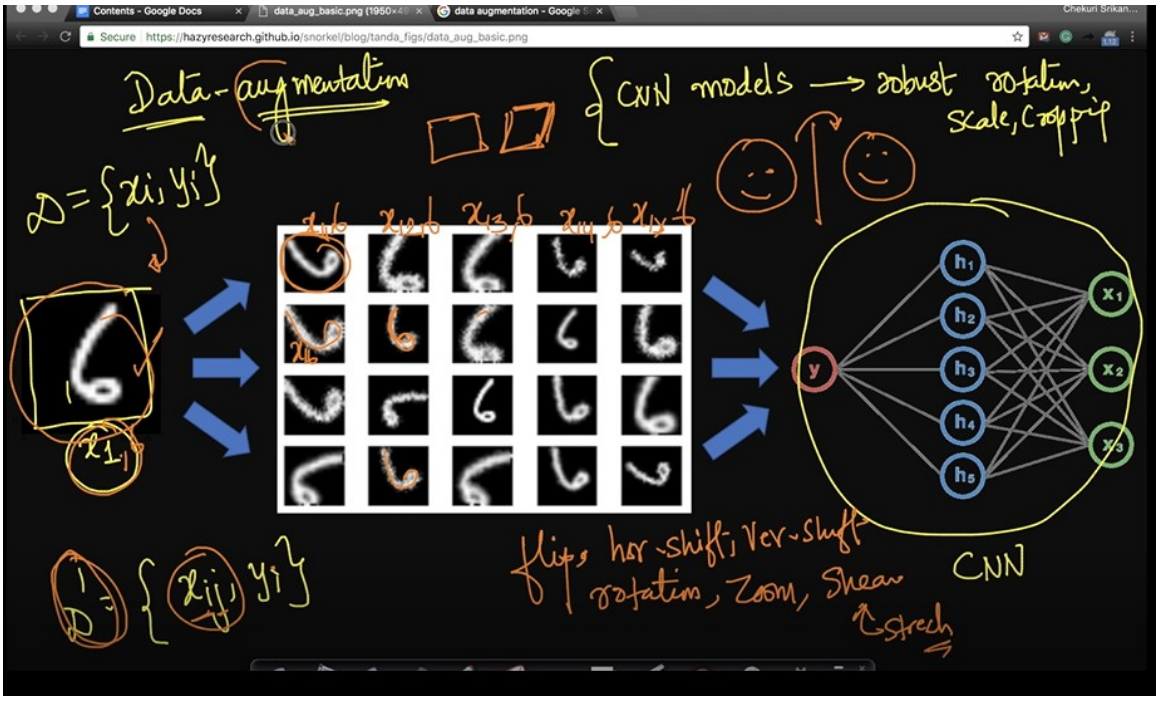
In 1998, they have the Sigmoid activation's.

Image Net:

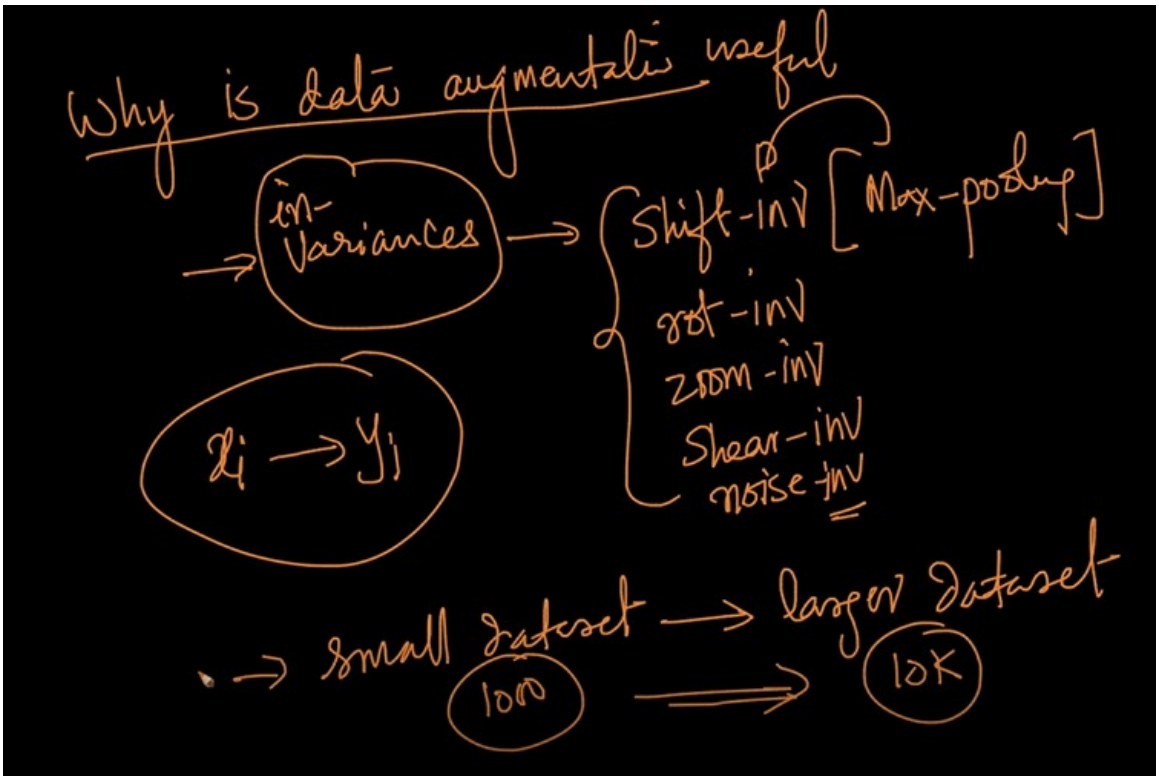
There are algorithmic improvements and computational power.

Data augmentation:

At each xi input we make multiple images, by rotating, adding noise, flipping, horizontal shift, vertical shift, zoom operation, shearing.



The data augmentation is useful because we get all the needed in variances that can be captured by pooling layers.



Since we are using the in variances, we get all the possible images with in variances. This is very popular especially in images.

Convolution layers in Keras:

This code is inspired by LeNet.

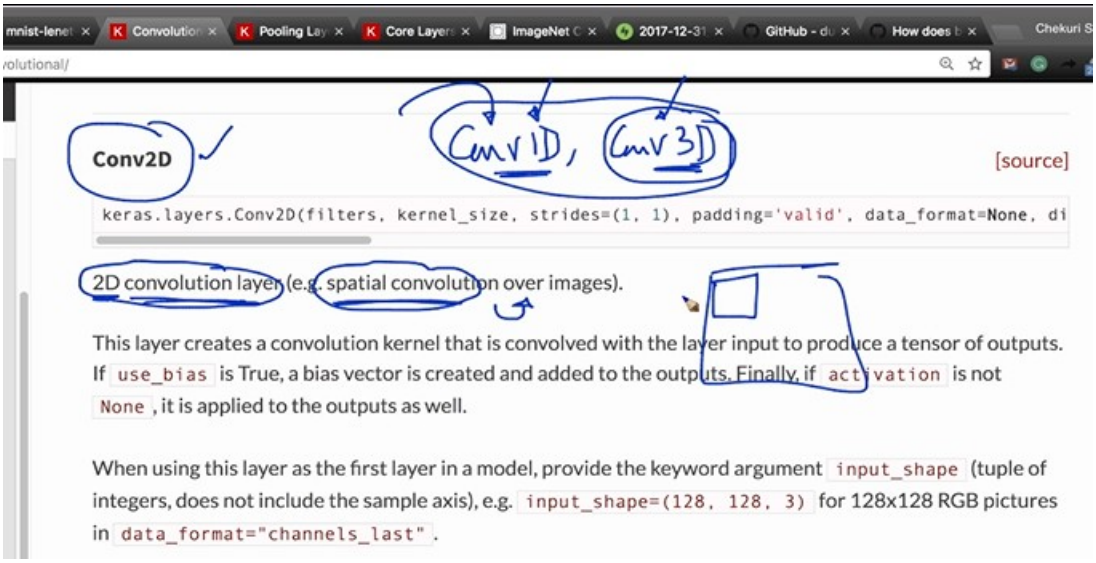
GitHub, Inc. [US] https://github.com/f00-/mnist-lenet-keras/blob/master/lenet.py

```python
class LeNet:
    @staticmethod
    def build(width, height, depth, classes, weightsPath=None):
        # initialize the model
        model = Sequential()

        # first set of CONV => RELU => POOL
        model.add(Convolution2D(20, 5, 5, border_mode="same",
            input_shape=(depth, height, width)))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # second set of CONV => RELU => POOL
        model.add(Convolution2D(50, 5, 5, border_mode="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # if weightsPath is specified load the weights
        if weightsPath is not None:
            model.load_weights(weightsPath)
```

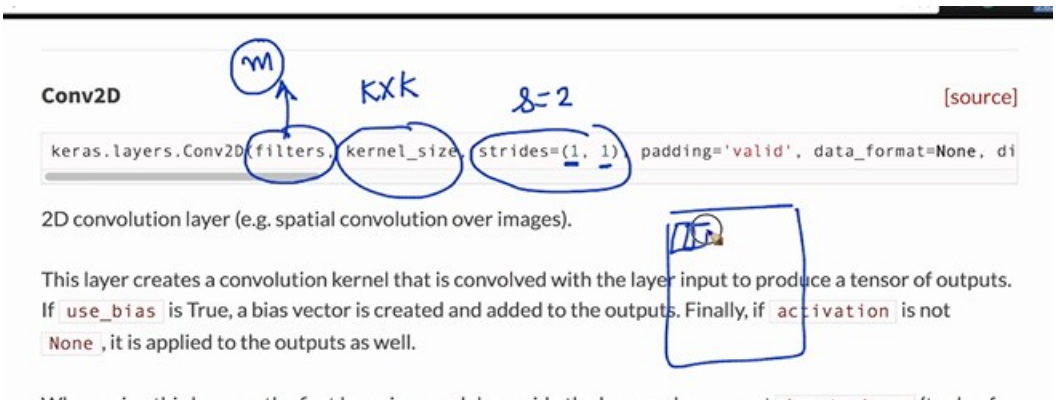
Sigmoid

Conv2D is in the space.

Conv:



Stride:



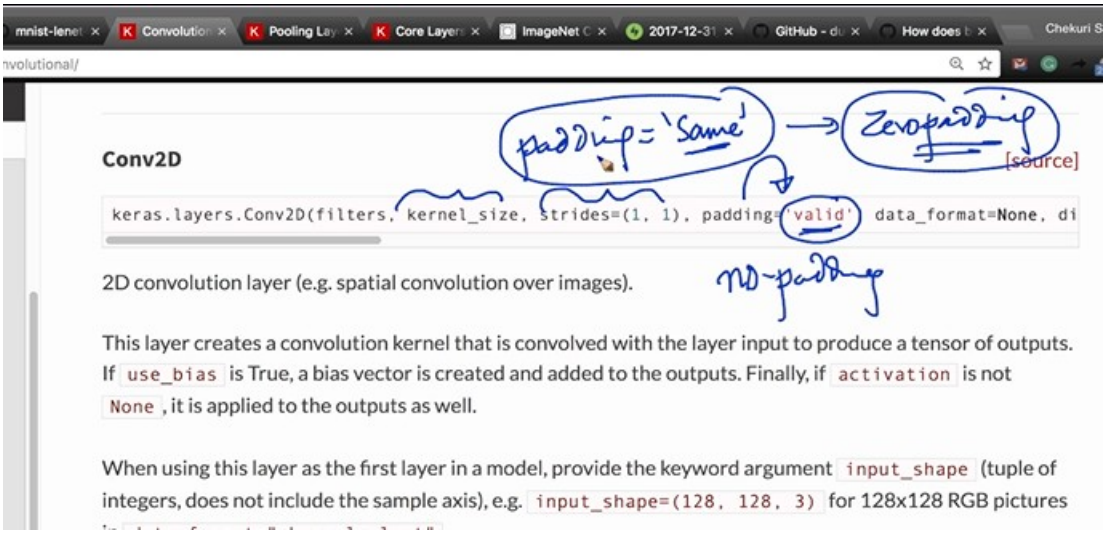
Stride happens on x and y axes, we can give the values explicitly.

padding:

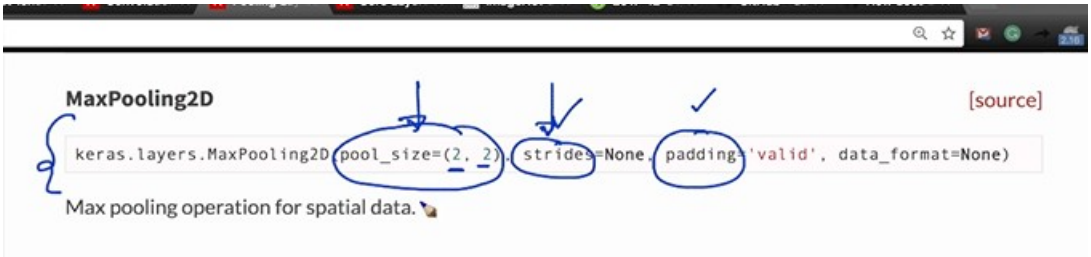
padding has "Valid" and "Same".

"Valid" - This gives no padding.

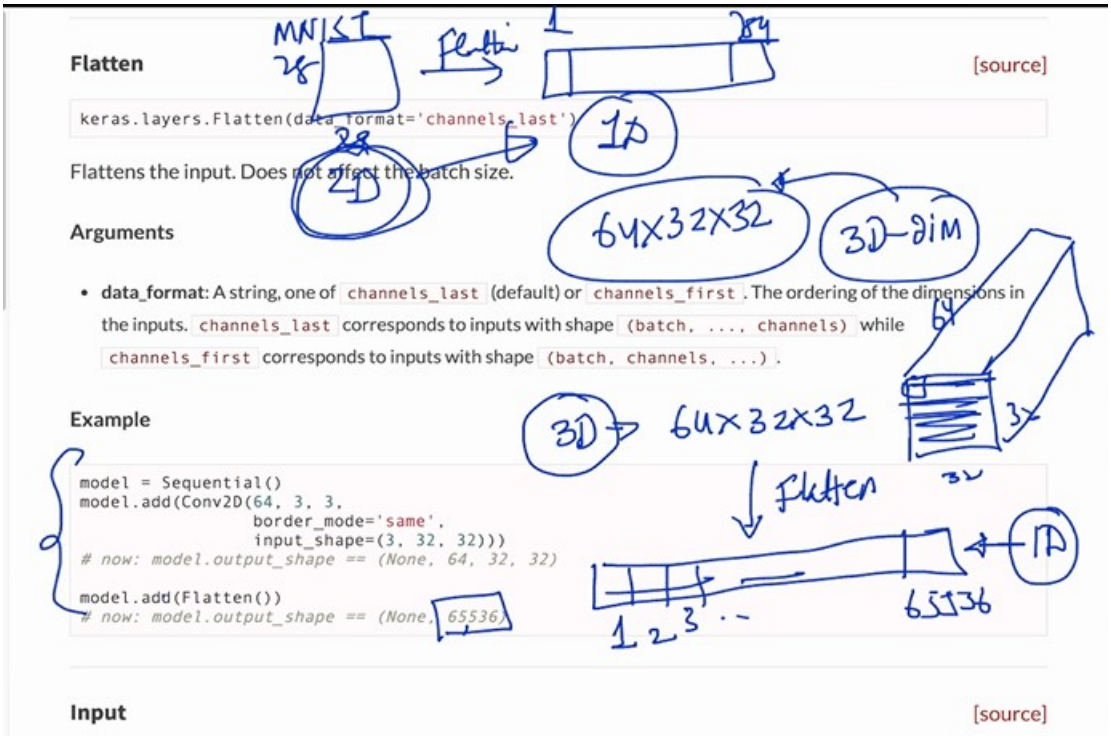
"Same" - This gives the same dimensions of the image as the input even after the conv layer.



max_pooling:



Flatten:

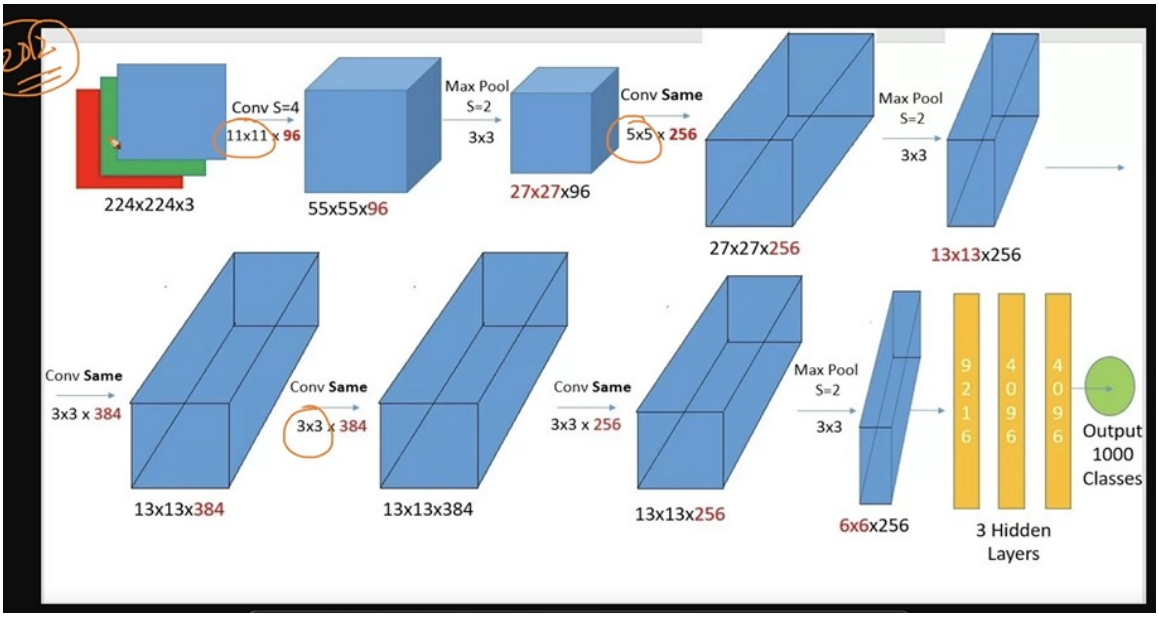


It takes the 3d tensor and gives the 1d array.

it is expected to give the input shape, especially for the first layer. It acts like a place holder.

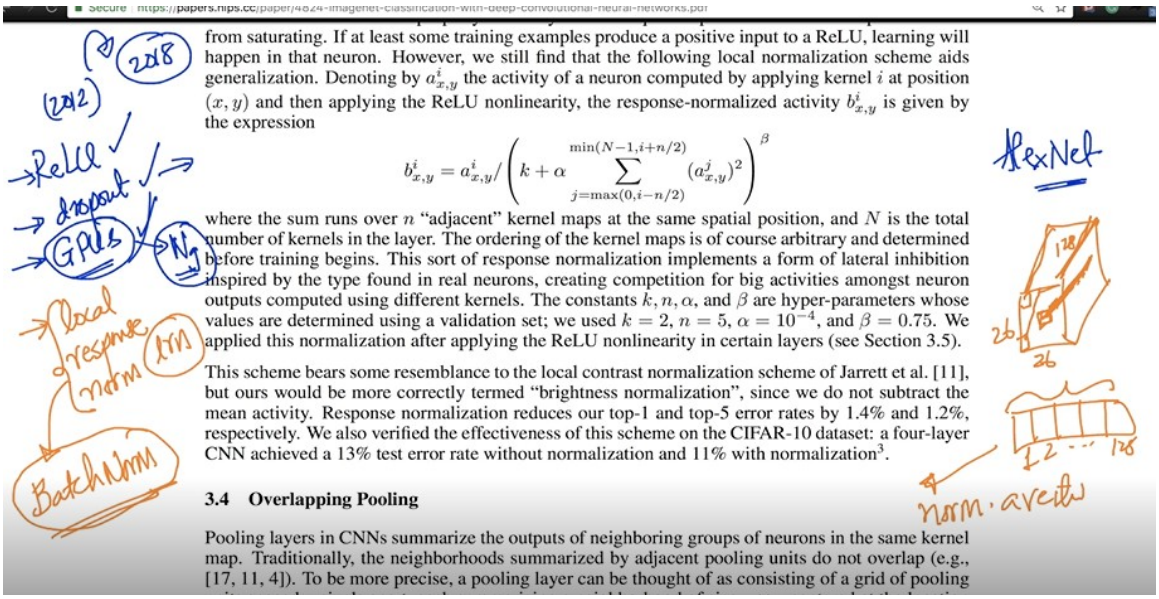
Alexnet:

we can think the output of the conv layer to have a depth equal to number of kernels.



The activation's that are used to be ReLU and dropouts(thees are applied at the fully connected layers), GPU's are used.

Local response normalization:



from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that the following local normalization scheme aids generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

where the sum runs over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$. We applied this normalization after applying the ReLU nonlinearity in certain layers (see Section 3.5).

This scheme bears some resemblance to the local contrast normalization scheme of Jarrett et al. [11], but ours would be more correctly termed "brightness normalization", since we do not subtract the mean activity. Response normalization reduces our top-1 and top-5 error rates by 1.4% and 1.2%, respectively. We also verified the effectiveness of this scheme on the CIFAR-10 dataset: a four-layer CNN achieved a 13% test error rate without normalization and 11% with normalization[3].

3.4 Overlapping Pooling

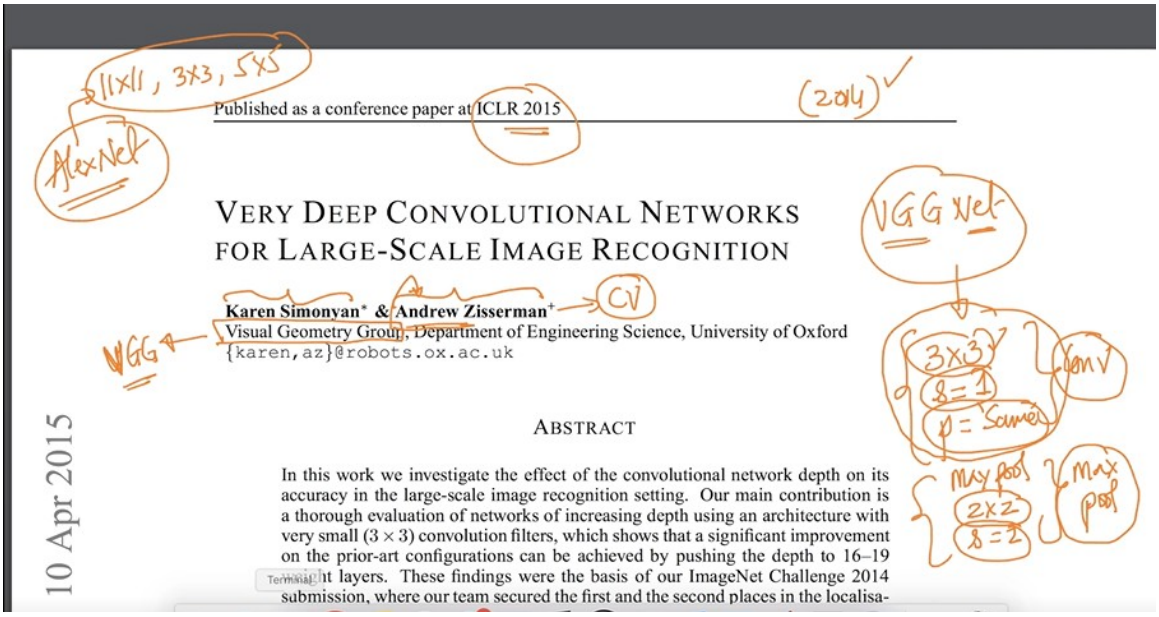
Pooling layers in CNNs summarize the outputs of neighboring groups of neurons in the same kernel map. Traditionally, the neighborhoods summarized by adjacent pooling units do not overlap (e.g., [17, 11, 4]). To be more precise, a pooling layer can be thought of as consisting of a grid of pooling

This Local normalization is done on the rows.

Vggnet:

It is a better version of AlexNet. We will use the kernel(3*3), stride(1) and padding('same').

We use the Max pooling of (2*2) with stride 2.



Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

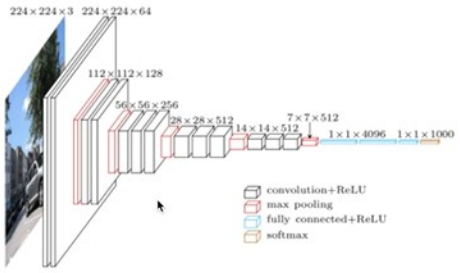
ABSTRACT

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3 × 3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 [illegible] layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisa-

VGG-16 is 16 layer network and VGG-19 is 19 layer network.

Quora Home Answer 3 Notifications 9 Search Quora 3 Add Question

Full view at image level:



Sources:

- Stack Overflow - Where Developers Learn, Share, & Build Careers
- VGG in TensorFlow

10.3k Views · View Upvoters

Upvote · 12 Downvote

VGG net:

GitHub, Inc. [US] | https://github.com/keras-team/keras/blob/master/keras/applications/vgg16.py

```
113         else:
114             img_input = input_tensor
115     # Block 1
116     x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv1')(img_input)
117     x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x)
118     x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)
119
120     # Block 2
121     x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv1')(x)
122     x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2_conv2')(x)
123     x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)
124
125     # Block 3
126     x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv1')(x)
127     x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv2')(x)
128     x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3_conv3')(x)
129     x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
130
131     # Block 4
132     x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv1')(x)
133     x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv2')(x)
134     x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4_conv3')(x)
135     x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)
136
137     # Block 5
138     x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv1')(x)
139     x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv2')(x)
140     x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5_conv3')(x)
141     x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
142
```

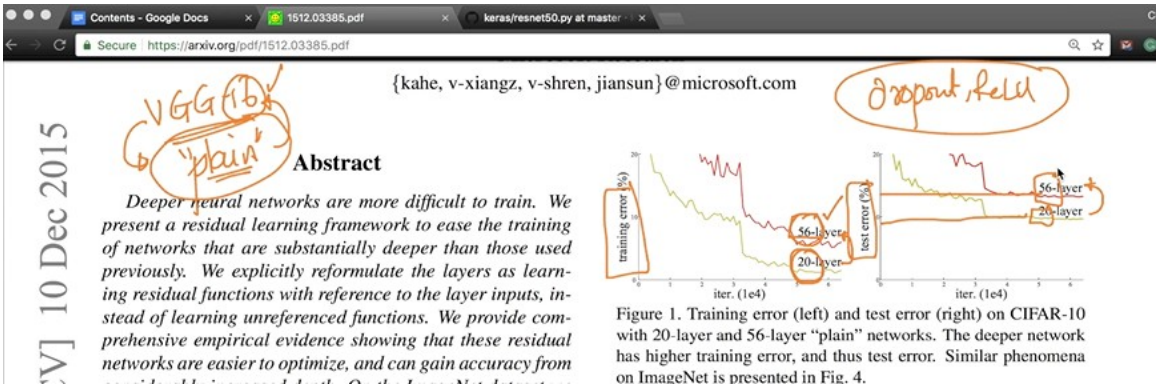
We will get the trained VGG model in the keras.

ResNET: Deep Residual learning networks for Image recognition.

Residual networks:

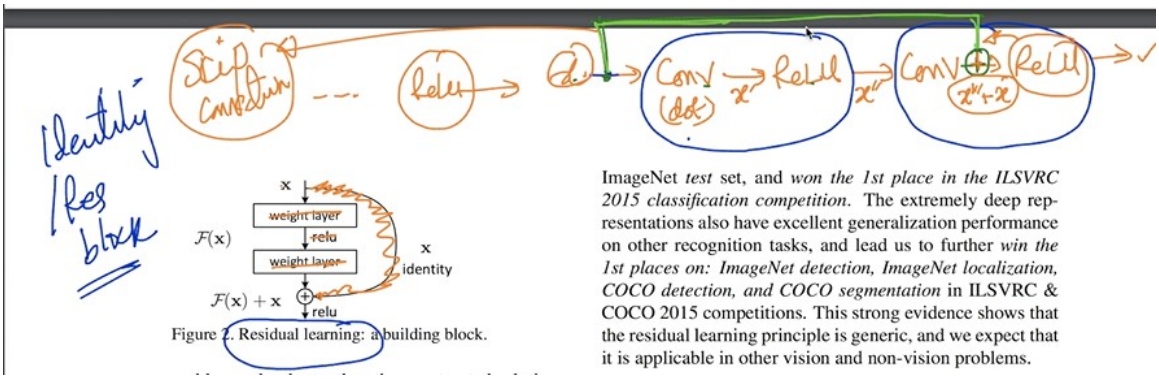
They found that the deeper networks are showing the more training and test errors than less deeper networks. To solve this problem ResNET's were proposed.

These are conv nets. The concepts like dropouts and relu are not useful.



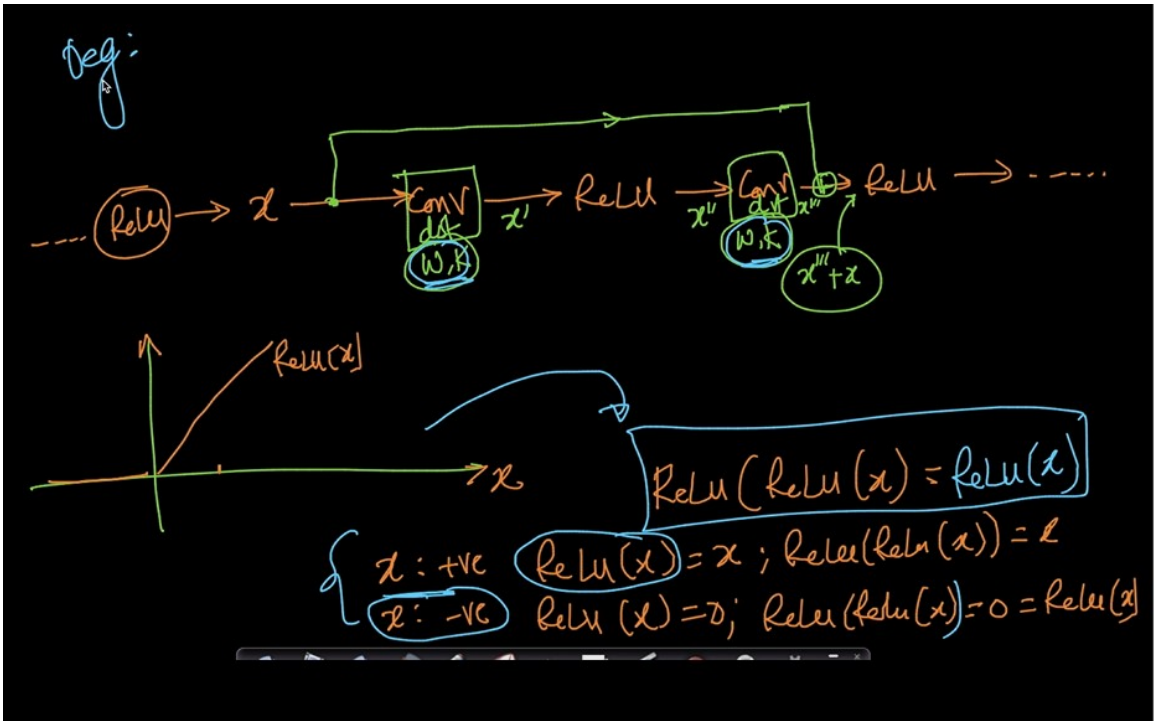
The residual block (or) identity block.

Here we use the skip connection, at some point in the network. This happens before the second ReLU.



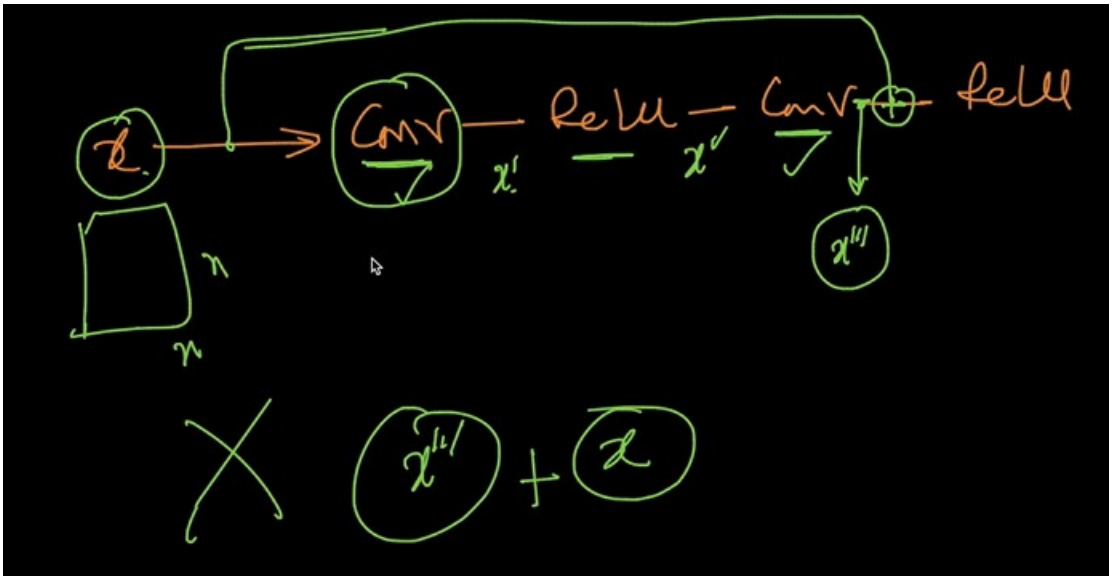
The addition happens before the Relu connection in the series.

These connections are called the skip connections.

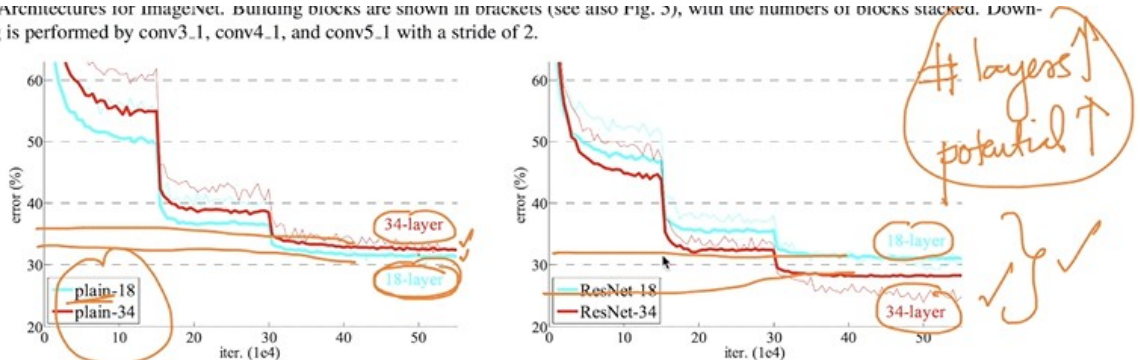


If some of these layers are useless, they are skipped due to some sort of regularization.

The convolutions that are performed here are of the same shape.



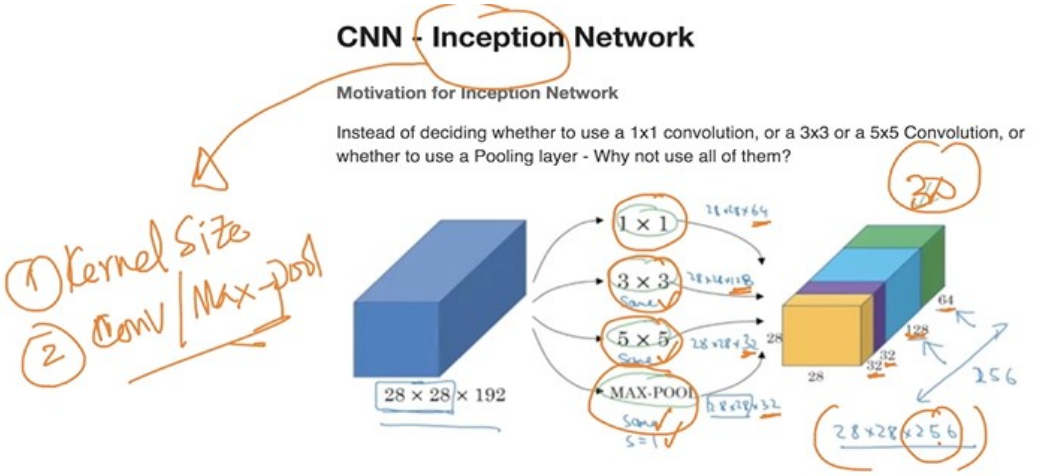
. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-
ıg is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.



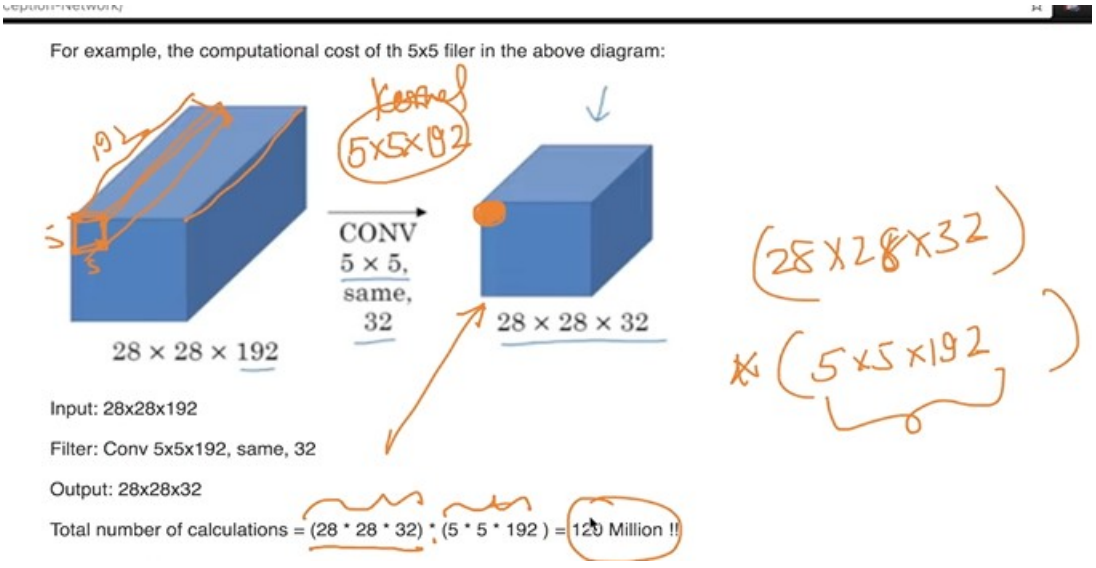
Resnet50 Keras implementation:

Inception network:

We use all the kernels in the inception network.

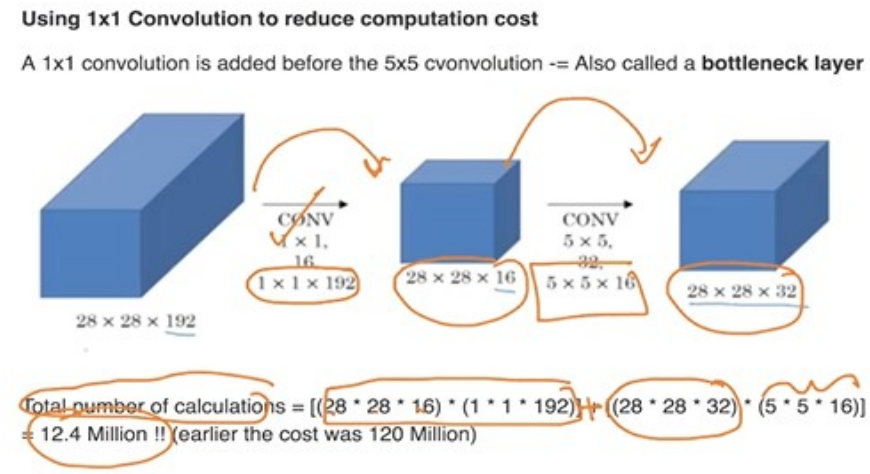


Here we get 120 million multiplications and additions.



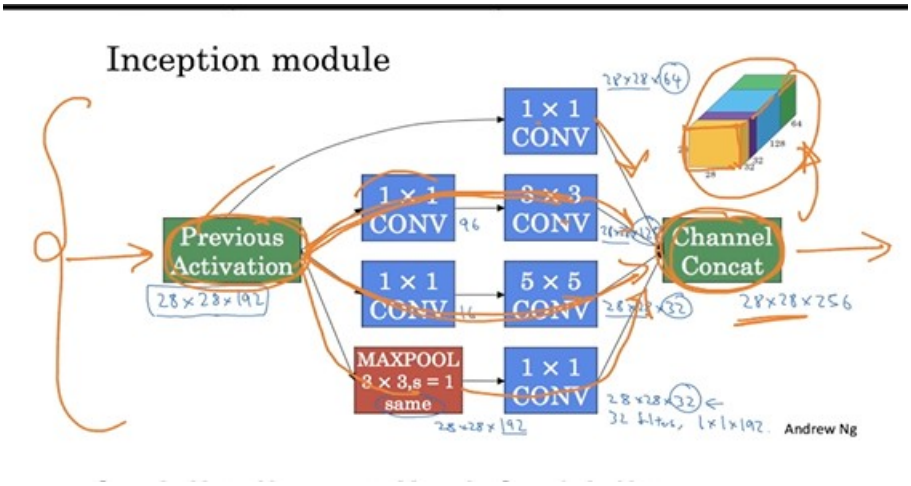
For only 5*5 convolution.

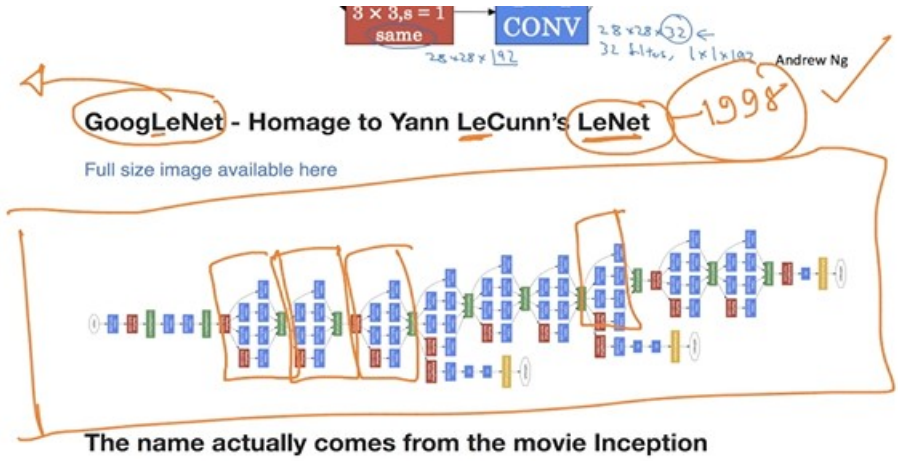
We will optimize this algo with 1*1 kernel and perform 5*5 conv, Then we get the same output as the 5*5 kernel.



Not we have only 12.4 million computations. This is a very nice engineering hack.

One inception module calculation:





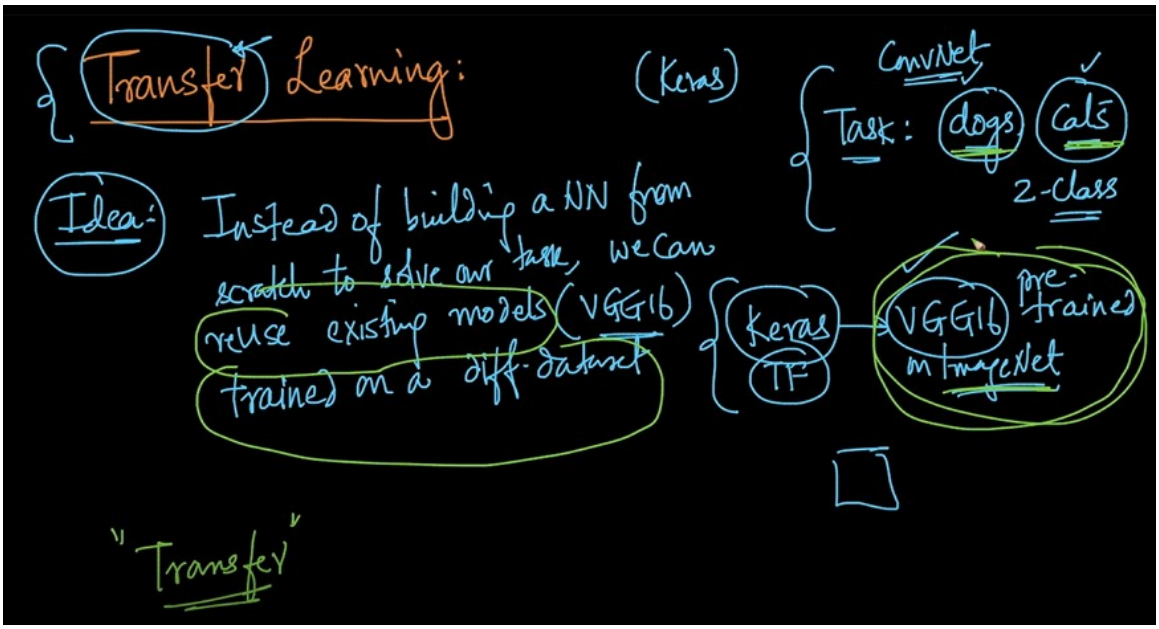
Transfer Learning:

Task: We want to build the conv neural network to classify the cats and dogs.

Instead of building the Conv layer, we can use the existing model architecture on a different data set.

These are called pre-trained model.

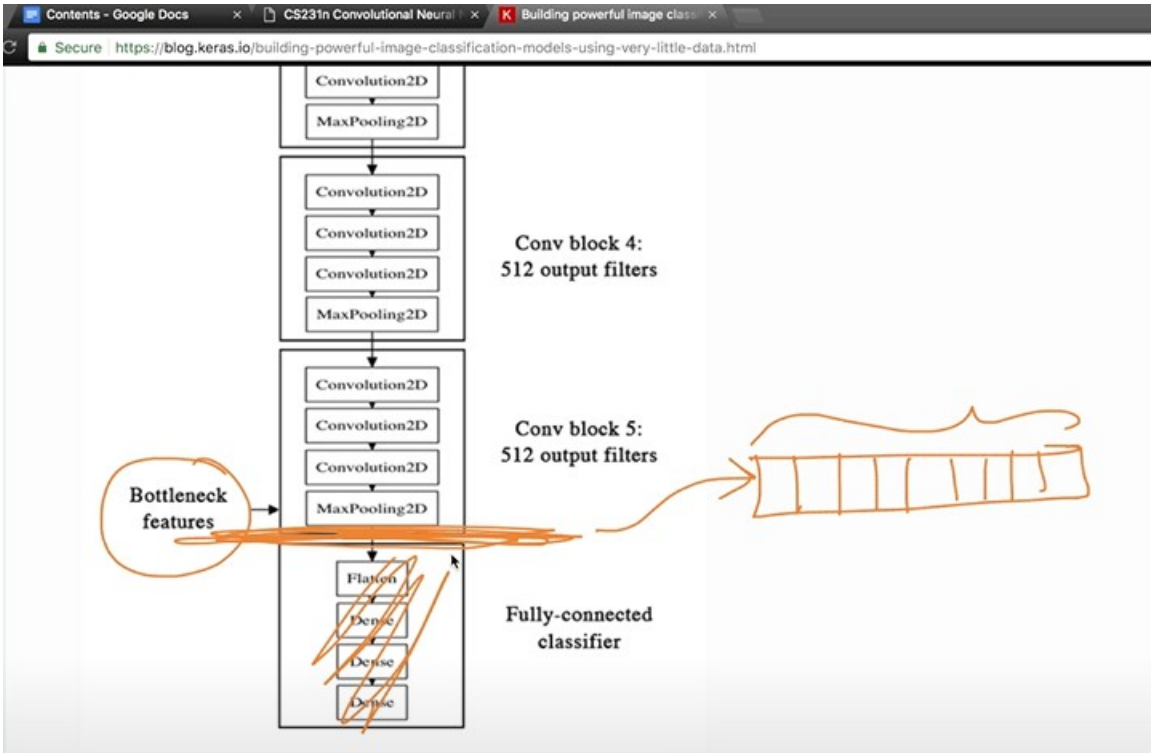
We will transfer a model from one data-set to the other data set, this is called the transfer learning.



Multiple cases in transfer learning.

Case – 1:

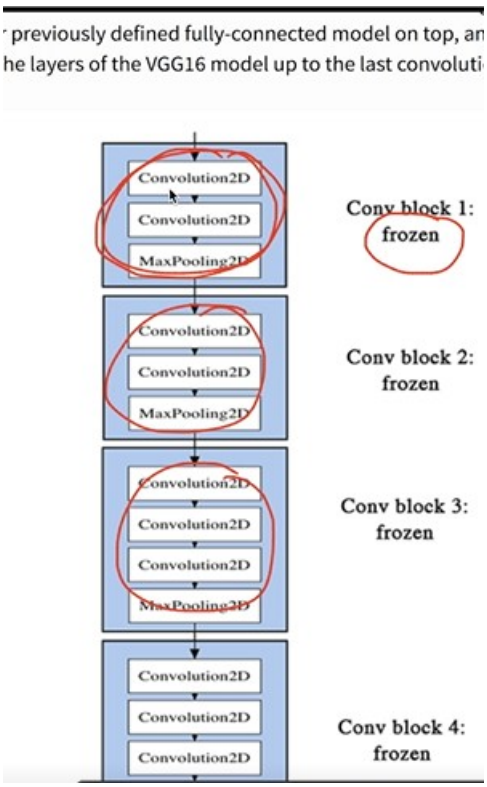
We learn to the last flatten layer and take the outputs, these features are called the bottle neck features.



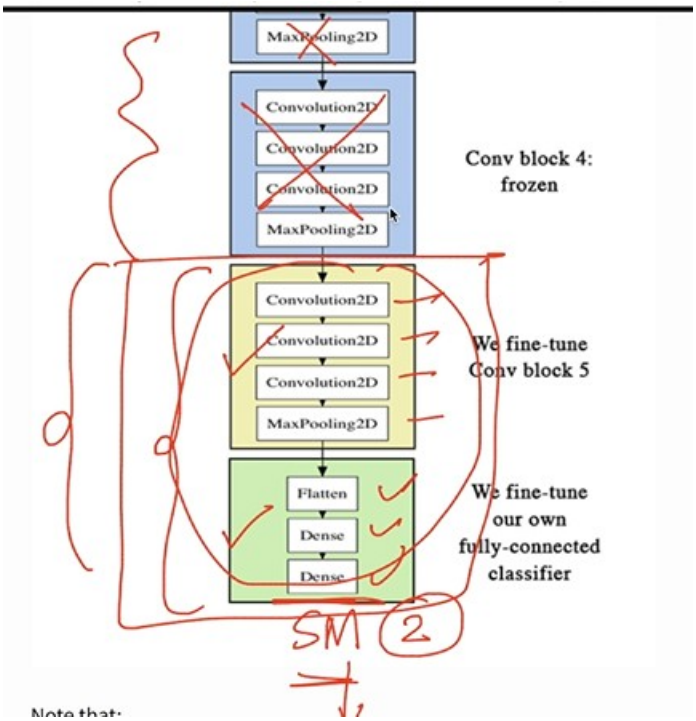
We use all the conv layers. Now,we train a simpler model like a logistic layer. This **is done just before the flatten layer but not after the flatten layer.**

Case – 2:

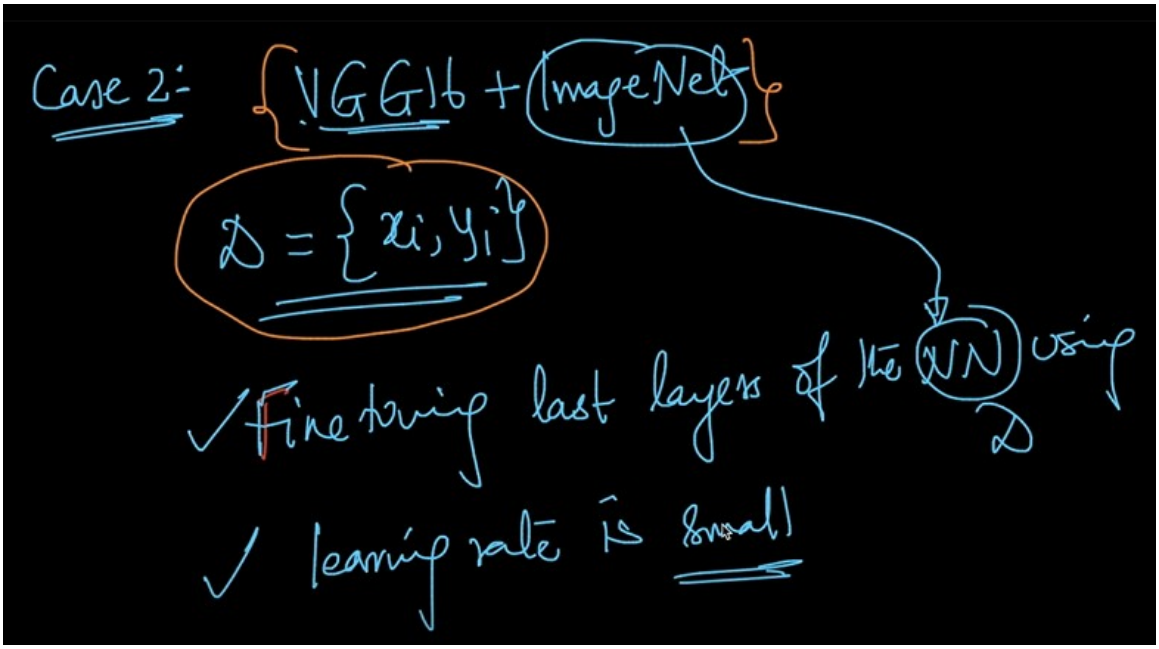
The earlier layer does the thing that is useful for all the image recognition tasks. We will freeze all the initial layers.



Now we fine tune the last layers and fine tune our won fully connected classifiers. I will train this whole network we do two class soft-max.



We are only fine tuning the last layers of the neural network, using the new data set. **One thing we have to ensure that the learning rate is small, so that the weights do not change drastically.**

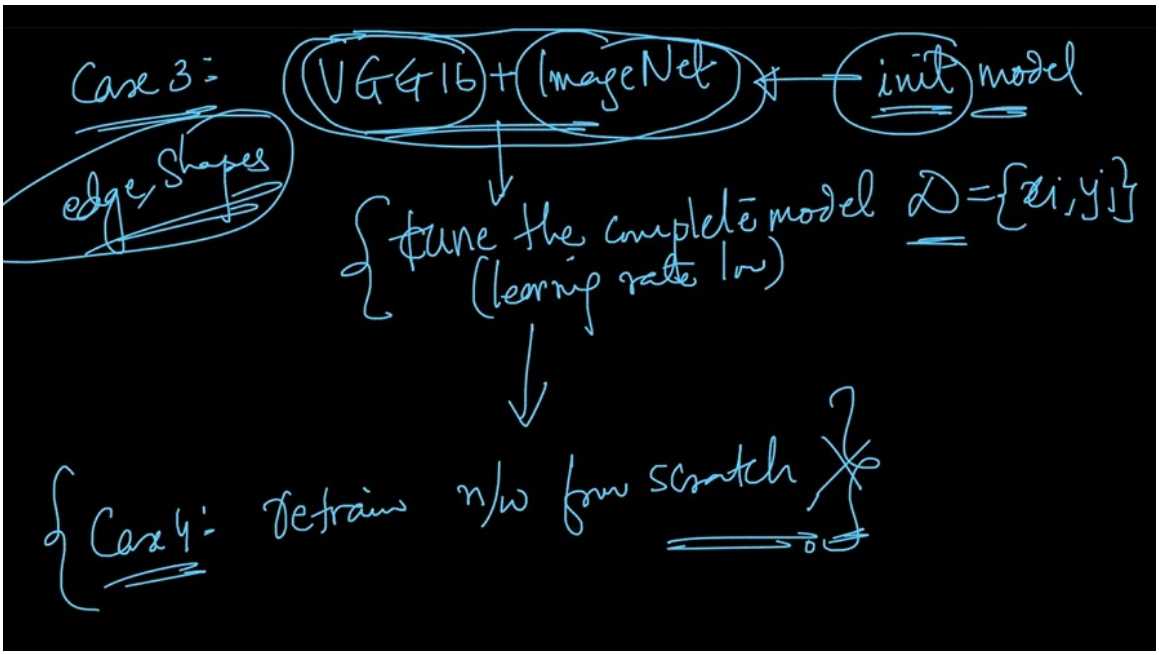


Case 3:

There is a third case, we have the model with vgg16 as the initialize the model, **and fine tune the whole model using the new data set.**

Case 4:

We can retrain the whole model from scratch.



When to use the different cases of transfer learning:

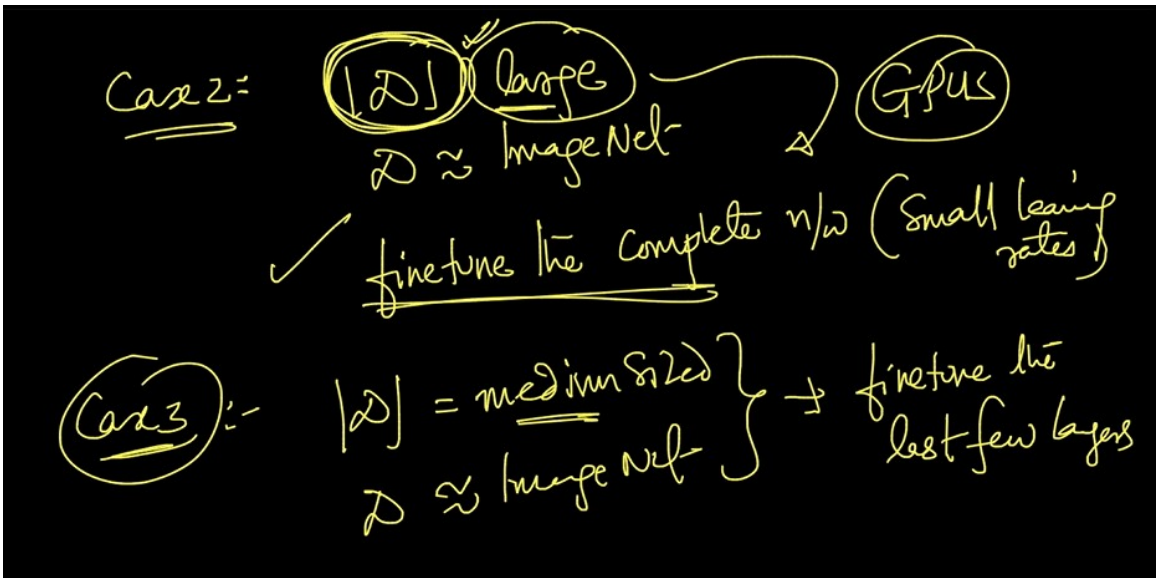
1. Depends on size of the task data set.

2. How different the data set to the image-net data set.

Case 1: If data set is very similar to the image-net data set, then use the VGG net as the feature engineering tool and just train a simple linear model as Case1 in Transfer Learning.

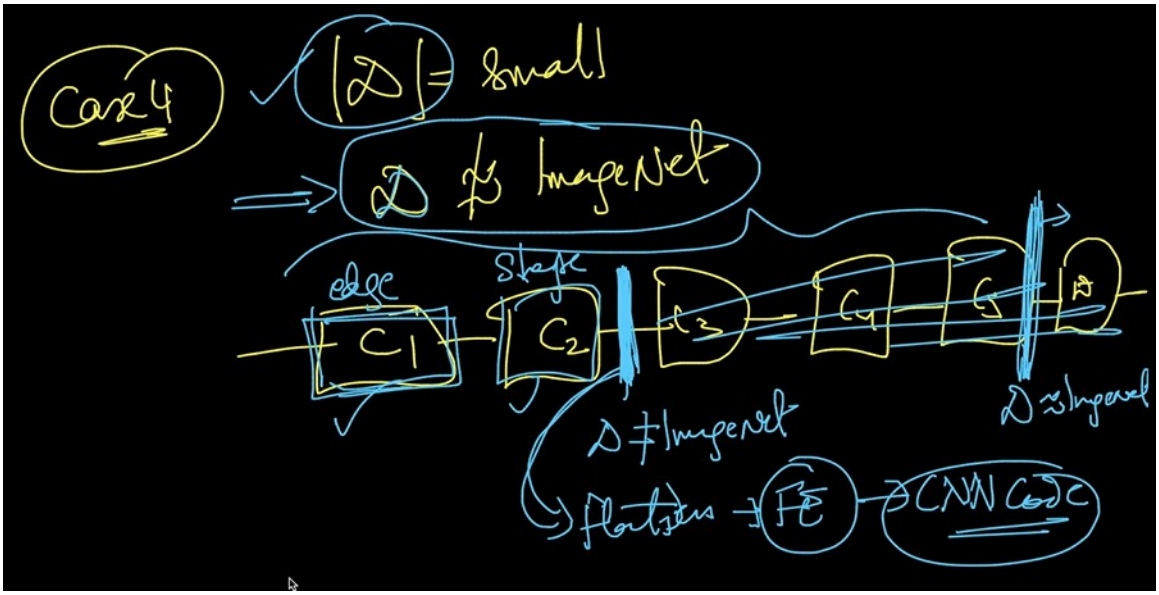
Case 2: Let's assume the dataset is large, then we can fine tune the complete network as Case3 in Transfer Learning. Unless we have GPU's.

Case 3: If the data set size is medium, then we will fine tune the last few layers



** Case 4: What is the data set is small and the data set is not same as the image net data set?

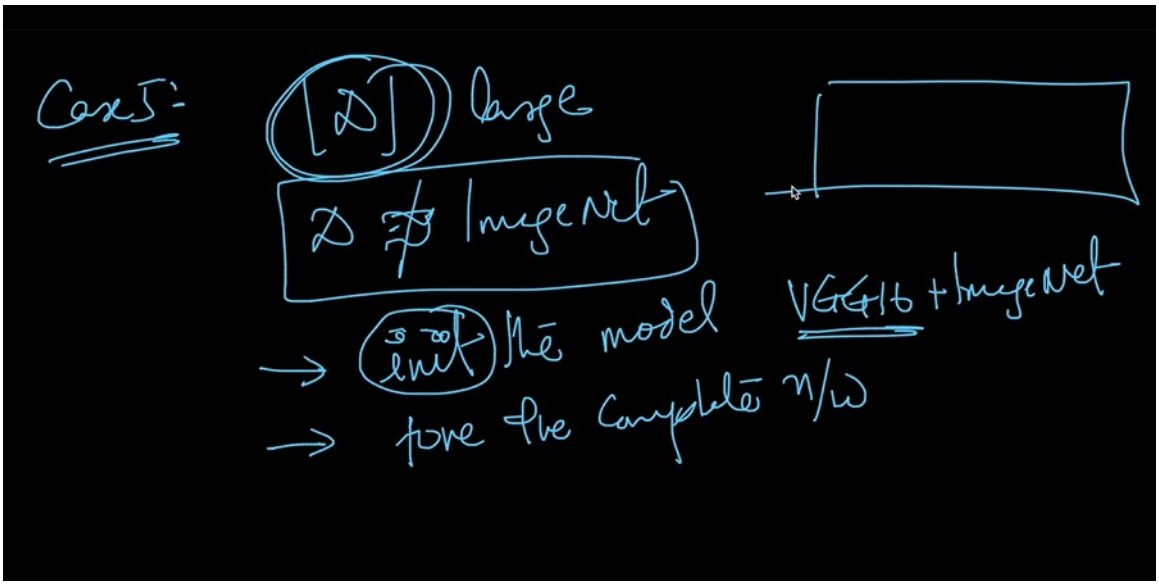
We use the earlier layers as the feature engineering steps and generate CNN Code, which is not similar to the image net.



Case – 5:

The data set is large and not similar to the Image net. We will initialize the model using the vgg16 trained on

image net and fine tune the complete network.



The whole veg model is well trained.

Transfer learning problem for Cats vs. Dogs.

Data Augmentation of the cats vs. dogs data set:

Let's look at an example right away:

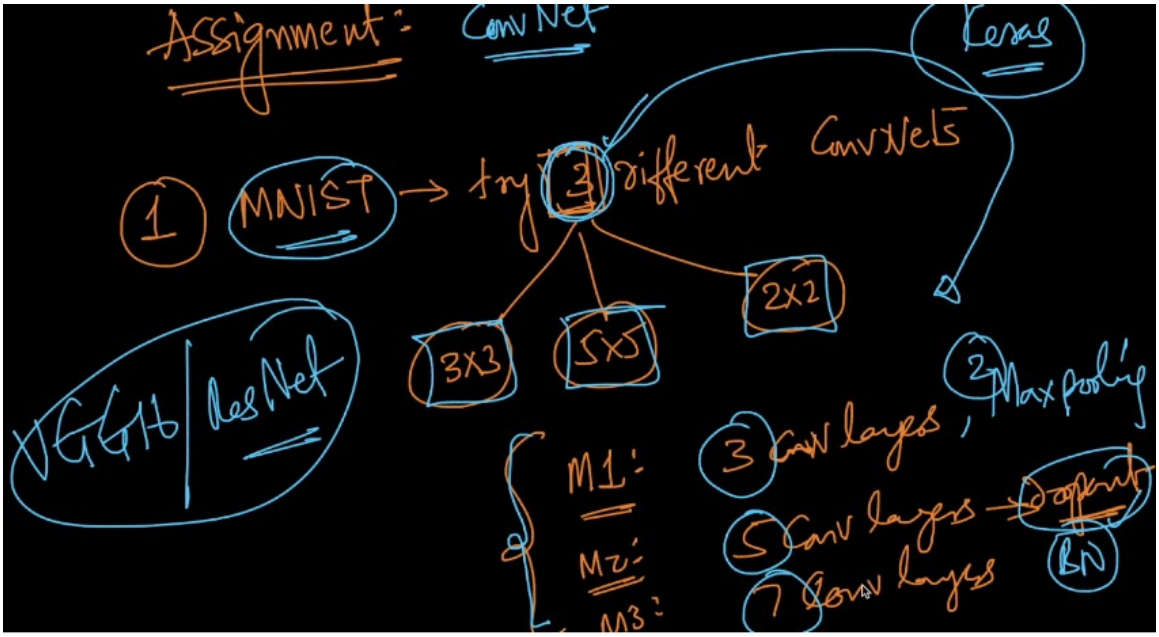
```
from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

We will create the whole model from scratch.

using bottleneck features:

adding the new layers to the con layers:

Assignment:



Create the 3 architectures of different